МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

С.Ю. ГАВРИЛЕНКО, А.М. КЛИМЕНКО,
Н.Ю. ЛЮБЧЕНКО, В.Г. СМОЛЯР, С.О. ТИШКО

ТЕОРІЯ ЦИФРОВИХ АВТОМАТІВ ТА ФОРМАЛЬНИХ МОВ. ВСТУПНИЙ КУРС

Харків 2011

МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

С.Ю. ГАВРИЛЕНКО, А.М. КЛИМЕНКО,
Н.Ю. ЛЮБЧЕНКО, В.Г. СМОЛЯР, С.О. ТИШКО

ТЕОРІЯ ЦИФРОВИХ АВТОМАТІВ ТА ФОРМАЛЬНИХ МОВ. ВСТУПНИЙ КУРС

Рекомендовано Міністерством освіти і науки України

як навчальний посібник для студентів вищих навчальних закладів,
що навчаються за освітніми напрямами
6.050102 “ Комп’ютерна інженерія” та 6.050101 “Комп’ютерні науки”

Харків НТУ “ХПІ” 2011

ББК 32.815
Т34
УДК 519.713:519.17:004.42

Рецензенти:

В.І. Слюсар, д-р техн. наук, професор, Заслужений діяч науки і техніки України, головний науковий співробітник, Центральний науково-дослідний інститут озброєння та військової техніки Збройних Сил України;

В.А. Краснобаєв, д-р техн. наук, професор , професор кафедри «Автоматизація та комп'ютерні технології», Харківський національний технічний університет сільського господарства імені Петра Василенка;

В.В. Мохор, д-р техн. наук, професор, головний науковий співробітник, Інститут проблем моделювання в енергетиці імені Г.Є. Пухова НАН України.

Гриф надано Міністерством освіти і науки України,
лист № 1/11-12217 від 29.12.2010 р.

Т34 Теорія цифрових автоматів та формальних мов. Вступний курс : навч. посібник / Гавриленко С. Ю., Клименко А. М., Любченко Н.Ю. та ін. – Харків : НТУ "ХПІ", 2011. – 176 с.

ISBN

Викладено питання задання та синтезу цифрових автоматів. Розглянуто особливості використання формальних мов і граматик для опису мов програмування та побудови компіляторів.

Призначено для студентів спеціальностей бакалавратів “Комп’ютерна інженерія” та “Комп’ютерні науки”.

Іл. 50. Табл. 61. Бібліогр. 25 назв.

ББК 32.815

ISBN

ВСТУП

Дослідження в галузі теорії автоматів розпочалися у 50-х роках минулого століття. Теорія автоматів є одним із фундаментальних блоків сучасної теоретичної та практичної інформатики. Модель скінченного автомата виявилася досить зручною у великій кількості додатків не тільки в інформатиці, але й в інших галузях інженерної діяльності.

Навчальний посібник відповідає програмам навчальних дисциплін «Прикладна теорія цифрових автоматів» для спеціальностей бакалаврату 6.050102 «Комп'ютерна інженерія» і «Методи та засоби комп'ютерних інформаційних технологій» для спеціальностей бакалаврату 6.050101 «Комп'ютерні науки». Він складається з 12 розділів, кожний з яких містить контрольні запитання. Кожен розділ посібника починається теоретичним матеріалом, який супроводжується великою кількістю прикладів, і завершується прикладами для самостійного рішення. Посібник може бути використаний викладачами для проведення практичних занять.

У першому розділі розглядаються питання теорії графів. Графи зустрічаються в різних задачах, тому алгоритми обробки графів дуже важливі. Теорія графів надає інженерові виключно зручний апарат для моделювання структурних властивостей систем і відношень між об'єктами найрізноманітнішої природи. Завдяки наочності і простоті цей апарат останнім часом набув широкого визнання і часто використовується при дослідженні та моделюванні різних систем. Теорія графів знаходить широке застосування в моделюванні інформаційних процесів, у програмуванні та у рішенні технічних задач. Вона дозволяє просто описувати складні явища і дає їм графічну інтерпретацію. Рисунок у вигляді графа дозволяє швидко зрозуміти проблему і на інтуїтивному рівні розробити раціональний алгоритм рішення.

У другому розділі розглядаються основні положення алгебри логіки. Цей розділ присвячений методам мінімізації булевих функцій. Знання даних методів корисне при вивченні, наприклад, таких розділів дискретної математики, як «Схеми з функціональних елементів» – для пониження складності схем і «Автоматні функції» – для довизначення частково визначених функцій. У розділі наведені елементи логіки висловів – булевої алгебри на множині {істина, хибність} та елементи логіки предикатів. Також дано визначення загальних понять нечіткої логіки, нечіткої алгебри та нечіткого числення.

У третьому, четвертому та п’ятому розділах розглянуті найбільш поширені булеві функції, основні закони булевої алгебри, подання булевих функцій у вигляді досконалої та мінімальної нормальних форм.

У шостому розділі наведено поняття абстрактного автомата та способи його задання.

Сьомий розділ присвячений синтезу структурного автомата, а у восьмому коротко наведено синтез схем на дешифраторах та мультиплексорах.

У дев’ятому розділі розглядається синтез автоматів Мілі та Мура за схемою алгоритму.

У десятому, одинадцятому та дванадцятому розділах описуються теорія формальних мов і граматик, їх використання при обробці текстів та побудові компіляторів мов програмування. Принципи і технології написання компіляторів настільки поширені, що можуть бути використані в різних галузях інформаційних технологій.

Автори висловлюють подяку всім фахівцям, завдяки спільним зусиллям яких ця книга побачила світ.

1. ГРАФИ. ОСНОВНІ ПОНЯТТЯ ТА ВИЗНАЧЕННЯ

Теорія графів надає виключно зручний апарат для моделювання структур різних систем і відношень між об'єктами найрізноманітнішої природи. Завдяки наочності і простоті цей апарат останнім часом завоював широке визнання і часто використовується при дослідженні та моделюванні різних інформаційних процесів.

1.1. Визначення графа

Багато завдань зводяться до розгляду сукупності об'єктів, істотні властивості яких описуються зв'язками між ними. Інтерес можуть представляти різні зв'язки і відношення між людьми, подіями, станами і взагалі між будь-якими об'єктами, представлені точками і сполучними лініями або стрілками. Такі схеми зустрічаються усюди під різними назвами: соціограми (у психології), симплекс (у топології), електричні ланцюги (у фізиці), діаграми організації (в економіці), мережі комунікацій, генеалогічні дерева і так далі. Д. Кенінг перший запропонував назвати такі схеми «графами». У подібних випадках зручно дані об'єкти зображати точками, які звуться *вершинами*, а зв'язки між ними – лініями (довільної конфігурації), які звуться *ребрами.*

Графи використовуються при аналізі і синтезі зі скінченним числом станів. Вершини графа в цьому випадку відповідають станам дискретної системи, а дуги, наприклад, умовам переходу між станами або вірогідності переходу між ними.

*Визначення***.** Множину вершин V, зв'язки між якими визначені множиною ребер E, називають графом і позначають $G = (V, E)$.

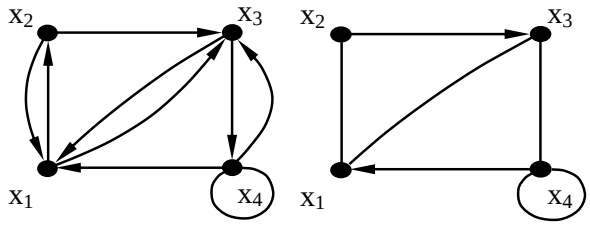
Часто зв'язки між об'єктами характеризуються цілком певною орієнтацією. Наприклад, на деяких вулицях допускається тільки односторонній автомобільний рух, транспортування газу газопроводом може бути направлене тільки в один бік, стосунки між людьми можуть визначатися підлеглістю або старшинством і тому подібне Орієнтовані зв'язки характеризують перехід системи з одного стану в інший, результати зустрічей між командами в спортивних змаганнях, різні відношення між числами (нерівність, подільність).

Для указання напряму зв'язку між вершинами графа відповідне ребро позначається стрілкою. Орієнтоване таким чином ребро називають *дугою*, а граф з орієнтованими ребрами – *орієнтованим графом* або коротше *орграфом* (рис. 1.1, *а*).

Якщо пара вершин з'єднується двома або більшою кількістю дуг, то такі дуги називають *паралельними*. При цьому дві дуги, однаково направлені по відношенню до даної вершини, називають *строго паралельними*, а різнонаправлені – *нестрого паралельними*. Нестрого паралельні дуги, що відображають орієнтацію зв'язку в обох напрямах, еквівалентні неорієнтованому зв'язку і можуть бути замінені ребром. Провівши таку заміну

в орграфі, отримаємо *змішаний* граф, який містить ребра і дуги (рис. 1.1, *б*). Будь-який неорієнтований або змішаний граф можна перетворити в орієнтований заміною кожного ребра парою нестрого паралельних дуг.

Подальше узагальнення відображення зв'язків між об'єктами за допомогою графів полягає в приписуванні ребрам і дугам деяких кількісних значень, якісних ознак або характерних властивостей, які називаються *вагами*. Граф, у якого задана вага для кожного ребра або дуги, називається *зваженим*. У простому випадку це може бути порядкова нумерація ребер і дуг, яка вказує на черговість при їх розгляді. Вага ребра або дуги може означати довжину (шляхи сполучення), пропускну здатність (трубопроводи), кількість набраних очок (турніри), кількість смуг руху (автомобільні дороги), характер стосунків між людьми (начальники, підлеглі) і тому подібне.



а *б*

Рисунок 1.1 – Орієнтований (*а*) і змішаний (*б*) графи

Вагу можна приписувати не тільки ребрам і дугам, але і вершинам. Наприклад, вершини, що відповідають населеним пунктам на карті автомобільних доріг, можуть характеризуватися кількістю місць в кемпінгах, пропускною здатністю станцій техобслуговування. Взагалі, вага вершини означає будь-яку характеристику відповідного їй об'єкта (колір предмета, що зображається вершиною, вік людини і т. ін.).

1.2. Типи скінченних графів

Визначення. Якщо множина вершин графа скінченна, то він називається скінченним графом. Скінченний граф $G = (V, E)$, що містить рівно p вершин і q ребер, називається (p, q) - графом.

Нехай $V = V = \{v_1, v_2, ... v_p\}$ і $E = \{e_1, e_2, ...e_q\}$ – відповідно, множина вершин і ребер (p, q)-графа. Кожне ребро $e_k \in E$ з'єднує пару вершин $v_i, v_j \in V$, що є його *кінцями* (*граничними вершинами*). Для орієнтованого ребра (дуги) розрізняють *початкову вершину*, з якої дуга виходить, і *кінцеву вершину*, в яку дуга заходить.

Ребро, граничними вершинами якого є одна і та ж вершина, називається *петлею*.

Ребра з однаковими граничними вершинами є паралельними і називаються *кратними*.

У загальному випадку граф може містити й *ізольовані вершини*, які не є кінцями ребер і не сполучені ні між собою, ні з іншими вершинами.

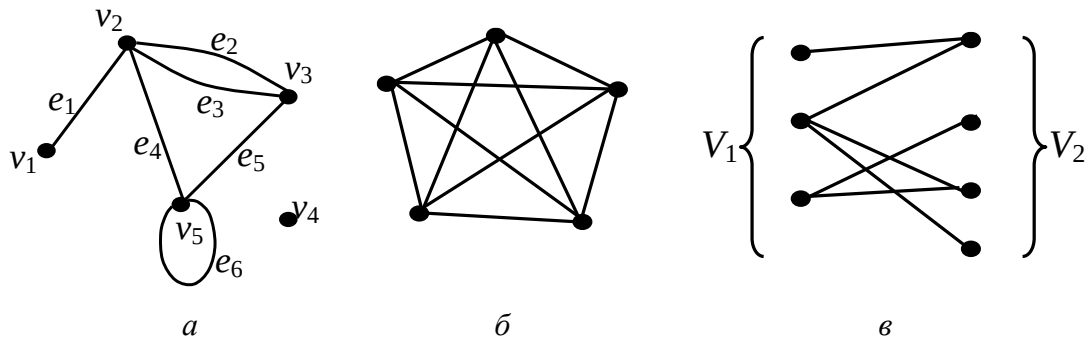
Кількість ребер, пов'язаних з вершиною v_i, (петля враховується двічі), називають *степенем вершини* і позначають через $\delta(v_i)$.

Степінь ізольованої вершини дорівнює нулю. Вершина степеня одиниці називається *кінцевою або висячою вершиною*.

Легко показати, що в будь-якому графі сума степенів всіх вершин дорівнює подвоєному числу ребер, а число вершин непарного степеня завжди парне.

В орграфі розрізняють додатні $\delta^+(v_i)$ і від'ємні $\delta^-(v_i)$ степені вершин, які дорівнюють відповідно числу витікаючих з v_i і таких, що заходять в v_i дуг. Отже, суми додатних і від'ємних ступенів всіх вершин орграфа рівні між собою і дорівнюють числу всіх дуг.

Приклад. Для скінченного (5, 6)-графа на рис.1.2, *а* множина вершин $V = \{v_1, v_2, v_3, v_4, v_5\}$; множина ребер $E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$; ребра e_2 та e_3 паралельні; ребро e_6 є петлею; v_4 – ізольована вершина і її степінь дорівнює 0 ($\delta(v_4) = 0$); v_1 – висяча вершина і її степінь дорівнює 1 ($\delta(v_1) = 1$); решта вершин графа мають такі степені: $\delta(v_2) = 4$, $\delta(v_3) = 3$, $\delta(v_5) = 4$. Для вершини x_3 орграфа на рис. 1.1, *а* маємо додатний ($\delta^+(x_3) = 2$) і від'ємний ($\delta^-(x_3) = 3$) степені.



а *б* *в*

Рисунок 1.2 – Типи графів: *а* – псевдограф; *б* – повний граф; *в* – дводольний граф (біграф)

Визначення. Простий або звичайний граф – це граф без петель і кратних ребер. Повний граф – це простий граф, в якому будь-які дві вершини сполучені ребром. Мультиграф – це граф, що не містить петель, але який має кратні ребра. Псевдограф – це граф, що допускає петлі і кратні ребра (найбільш загальний випадок графа). Порожній або нуль-граф – це граф, що не має ребер ($E = \varnothing$, всі його вершини ізольовані).

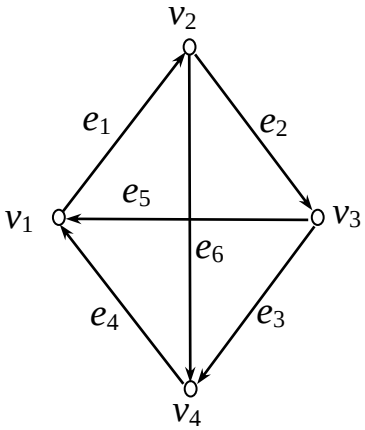


Рисунок 1.3 – Кубічний повний граф

Якщо множина вершин V простого графа допускає таке розбиття на дві непересічні підмножини V_1 і V_2 ($V_1 \cap V_2 = \varnothing$), що не мають ребер, які сполучають вершини однієї і тієї ж підмножини, то він називається *дводольним* або *біграфом*.

Орієнтований граф вважається *простим*, якщо він не має строго паралельних дуг і петель.

Граф, степені всіх вершин якого однакові і рівні r, називається *однорідним* (*регулярним) r*-го *степеня.* Повний граф з n вершинами завжди є однорідним графом степеня $n - 1$, а порожній граф – однорідний граф степеня 0. Граф третього степеня називають *кубічним графом* (рис. 1.3). Він володіє багатьма цікавими властивостями і, зокрема, завжди має парне число вершин.

Приклад 1. На рис. 1.2, *а* наведено псевдограф з петлею e_6 і кратними ребрами e_2 і e_3. Біграф з двома неперетинними підмножинами вершин V_1 і V_2, наведений на рис. 1.2, *в*, це простий граф. На рис. 1.2, *б* показаний повний граф степеня 4, а на рис. 1.3 – повний (орієнтований) кубічний граф. Орграф, показаний на рис. 1.1, *а*, не є простим.

1.3. Суміжність та інцидентність

Визначення. Дві вершини v_i і $v_j \in V$ графа $G = (V, E)$ називаються суміжними, якщо вони є граничними вершинами ребра $e_k \in E$.

Відношення суміжності на множині вершин графа можна визначити, представивши кожне ребро як пару суміжних вершин, тобто $e_k = (v_i, v_j)$, $k = 1, 2, \ldots q$. Для неорієнтованих графів такі пари невпорядковані, тобто $e_k = (v_i, v_j) = (v_j, v_i)$, а для орграфів – впорядковані, причому v_i і v_j означають відповідно початкову і кінцеву вершини дуги e_k. Петля при вершині v_i представляється парою (v_i, v_i). Множина вершин V разом з визначеним на ній відношенням суміжності повністю описує граф.

Граф можна представити *матрицею суміжності*. Рядки і стовпці цієї квадратної матриці відповідають вершинам графа, а її (ij)-елемент дорівнює числу кратних ребер, що сполучають вершини v_i і v_j (або направлені від вершини v_i до вершини v_j для орграфа).

Матриця суміжності неорієнтованого графа завжди симетрична, а орграфа – несиметрична. Неорієнтованим ребрам відповідають пари ненульових елементів, симетричних відносно головної діагоналі матриці, дугам – ненульові елементи матриці, а петлям – ненульові елементи головної діагоналі. У стовпцях і рядках, відповідних ізольованим вершинам, всі елементи дорівнюють нулю. Елементи матриці простого графа завжди дорівнюють 0 або 1, причому всі елементи головної діагоналі нульові.

Приклад 2. Графи, наведені на рис. 1.2, *а* і 1.3, описуються матрицями суміжності V_1 і V_2:

$V_1 =$

	v_1	v_2	v_3	v_4	v_5
v_1		1			
v_2	1		2		1
v_3		2			1
v_4					
v_5		1	1		1

$V_2 =$

	v_1	v_2	v_3	v_4
v_1		1		
v_2			1	1
v_3	1			1
v_4	1			

Для зваженого графа, що не містить кратних ребер, можна узагальнити матрицю суміжності так, що кожен її ненульовий елемент дорівнює вазі відповідного ребра або дуги. Будь-яка квадратна матриця n-го порядку може бути представлена орграфом з n вершинами, дуги якого сполучають суміжні вершини і мають ваги, рівні відповідним елементам матриці. Якщо матриця симетрична, то вона може бути представлена неорієнтованим графом.

Визначення. Якщо вершина v_i є кінцем ребра e_k,, то говорять, що вони інцидентні: вершина v_i інцидентна ребру e_k і ребро e_k інцидентне вершині v_i.

Тоді як суміжністю є відношення між однорідними об'єктами (вершинами), інцидентність – це відношення між різнорідними об'єктами (вершинами і ребрами). При розгляді орграфів розрізняють *додатну інцидентність* (дуга виходить з вершини) і *від'ємну інцидентність* (дуга заходить у вершину).

Для (p,q)-графа можна побудувати *матрицю інцидентності* розміру $p \times q$, рядки якої відповідають вершинам, а стовпці – ребрам. Для неорієнтованого графа елементи цієї матриці визначаються за таким правилом: *ij-елемент* дорівнює 1, якщо вершина v_i інцидентна ребру e_j, і дорівнює нулю, якщо v_i, і e_j не інцидентні. У випадку орграфа ненульовий *ij*-елемент дорівнює 1, якщо v_i початкова вершина дуги e_j, і дорівнює –1, якщо v_i – кінцева вершина дуги e_j.

Приклад 3. Матриці інцидентності графів, наведених на рис. 1.2, *а* і 1.3, мають відповідно такий вигляд:

$A_1 =$

e_1	e_2	e_3	e_4	e_5	e_6	
1						v_1
1	1	1	1			v_2
	1	1		1		v_3
						v_4
			1	1		v_5

;

$A_2 =$

e_1	e_2	e_3	e_4	e_5	e_6	
1			-1	-1		v_1
-1	1				1	v_2
	-1	1		1		v_3
		-1	1		-1	v_4

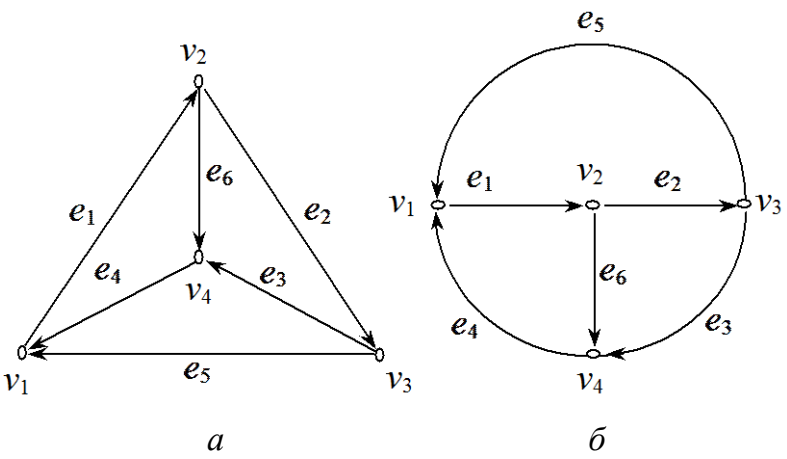
Кожен стовпець матриці інцидентності обов'язково містить два одиничні елементи (для орграфа ці елементи завжди мають різні знаки і дорівнюють відповідно 1 і –1). Кількість одиниць в рядку рівна степеню відповідної вершини (для орграфа кількість додатних одиниць визначає додатний степінь, а кількість від'ємних одиниць – від'ємний степінь). Нульовий рядок відповідає ізольованій вершині, а нульовий стовпець – петлі, причому нульовий стовпець матриці інцидентності лише вказує на наявність петлі, але не містить відомостей про те, з якою вершиною ця петля пов'язана (тобто матриця інцидентності неоднозначно визначає граф, проте в практичному застосуванні це може бути несуттєво).

*Визначення***.** Графи, для яких зберігається відношення інцидентності, називаються ізоморфними.

Приклад 4. Графи, зображені на рис. 1.4, мають таку ж матрицю інцидентності (A_2), як і граф рис. 1.3, проте з геометричної точки зору вони

абсолютно різні, хоча розрізняються лише зображенням, а відношення інцидентності (при відповідному позначенні вершин і ребер) однакові.

Зрозуміло, що матриця інцидентності визначає граф без петель з точністю до ізоморфізму. Зазвичай на її основі можна зобразити різні в геометричному відношенні, але ізоморфні між собою графи, кожен з яких називають *геометричною реалізацією.*



а *б*

Рисунок 1.4 – Ізоморфні графи

Якщо істотні властивості графа не пов'язані зі способом його зображення на площині або нумерацією вершин і ребер, то ізоморфні графи, як правило, не розрізняються між собою.

1.4. Способи задання графів

Геометричне подання – рисунок. Використовують з метою ілюстрації та як робочий документ (карти, схеми). Незручний з підвищенням кількості вершин.

Матриця суміжності графа (орграфа). Це квадратна матриця $\|\delta_{ij}\|$, елементи якої δ_{ij} дорівнюють одиниці лише, якщо вершина i є інцидентна початку дуги, а вершина j інцидентна кінцю дуги, інакше елементи дорівнюють нулю. Неорієнтований граф має матрицю, яка симетрична відносно головної діагоналі. Спосіб має обмеження – неможливо задавати мультиграфи.

Матриця інцидентності. Це прямокутна матриця, в якій розмітка вздовж стовпця – ідентифікатори вершин графа, розмітка вздовж рядка – ідентифікатори дуг або ребер. Елемент матриці $\|\varepsilon_{ij}\|$, тобто, ε_{ij}, де i – номер рядка (а також вершини) і j – номер стовпця (а також дуги або ребра), набуває значень:

– якщо вершина i є початок дуги i, то $\varepsilon_{ij} = 1$;

– якщо вершина i є кінець дуги i, то $\varepsilon_{ij} = -1$;

– якщо вершина i не є інцидентна дузі або ребру i, то $\varepsilon_{ij} = 0$.

У матриці інцидентності не більше двох елементів рядка є відмінні від нуля. Такий спосіб задання графів є універсальним, але не є економічним з точки зору використання пам'яті ЕОМ.

Список ребер (дуг). Тут у кожному рядку є відповідні номери дуги та вершин, що їй інцидентні. Це також універсальний, але набагато більш економічний спосіб з точки зору використання пам'яті ЕОМ.

Матриця досяжності орграфа. Це також квадратна матриця, де номери рядків та стовпців це номери вершин орграфа. Елементи матриці набувають значення одиниці лише тоді, коли існує шлях з вершини, що має номер рядка, у вершину, що має номер стовпця матриці. Цей спосіб задання не дозволяє розрізнити зв'язні графи. Для кожного орграфа можна задати матрицю досяжності. Але не всяка довільна матриця є матрицею досяжності орграфа, бо відшукати з її допомогою інші варіанти задання іноді неможливо (орграф не існує).

Приклад 5. Для орграфа *G* (рис. 1.5) приготувати задання різними способами.

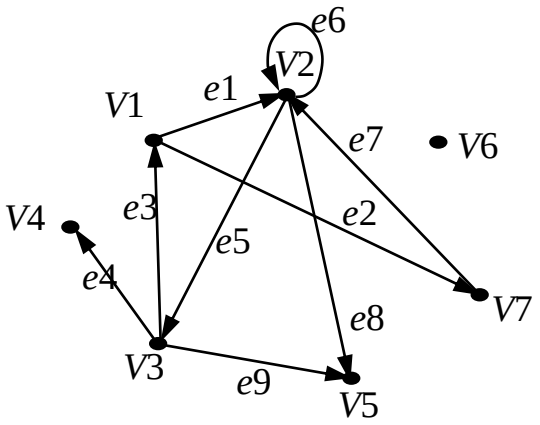


Рисунок 1.5 – Орграф *G*

1) Матриця інцидентності орграфа *G*:

	*e*1	*e*2	*e*3	*e*4	*e*5	*e*6	*e*7	*e*8	*e*9
*V*1	1	1	-1	0	0	0	0	0	0
*V*2	-1	0	0	0	1	0	-1	1	0
*V*3	0	0	1	1	-1	0	0	0	1
*V*4	0	0	0	-1	0	0	0	0	0
*V*5	0	0	0	0	0	0	0	-1	-1
*V*6	0	0	0	0	0	0	0	0	0
*V*7	0	-1	0	0	0	0	1	0	0

2) Матриця суміжності орграфа *G*:

	*V*1	*V*2	*V*3	*V*4	*V*5	*V*6	*V*7
*V*1	0	1	0	0	0	0	1
*V*2	0	1	1	0	1	0	0
*V*3	1	0	0	1	1	0	0
*V*4	0	0	0	0	0	0	0
*V*5	0	0	0	0	0	0	0
*V*6	0	0	0	0	0	0	0
*V*7	0	1	0	0	0	0	0

Задання мультиграфа матрицею інцидентності не є проблемним, а ось ізольована вершина тут ніяк не позначена.

Кількість одиниць у матриці суміжності дорівнює кількості дуг орграфа.

3) Список дуг:

*e*1	*e*2	*e*3	*e*4	*e*5	*e*6	*e*7	*e*8	*e*9
*V*1	*V*1	*V*3	*V*3	*V*2	*V*2	*V*7	*V*2	*V*3
*V*2	*V*7	*V*1	*V*4	*V*3	*V*2	*V*2	*V*5	*V*5

4) Матриця досяжності орграфа G:

	$V1$	$V2$	$V3$	$V4$	$V5$	$V6$	$V7$
$V1$	1	1	1	1	1	0	1
$V2$	1	1	1	1	1	0	1
$V3$	1	1	1	1	1	0	1
$V4$	0	0	0	0	0	0	0
$V5$	0	0	0	0	0	0	0
$V6$	0	0	0	0	0	0	0
$V7$	1	1	1	1	1	0	1

Така матриця інформативна саме для орграфа, бо для зв'язного графа всі елементи матриці є одиницями, а для графа з декількома областями зв'язності у матриці буде декілька квадратних підматриць спільно з одиницями.

1.5. Маршрути і підграфи

Визначення. Маршрут довжини m на графі – це послідовність m ребер графа (не обов'язково різних), таких, що граничні вершини двох сусідніх ребер співпадають. Маршрут проходить і через всі вершини, які інцидентні вхідним у нього ребрам.

Замкнутий маршрут – це маршрут, який приводить в ту ж вершину, з якої він почався. Маршрут, всі ребра якого різні, називається *ланцюгом,* а маршрут, для якого різні всі вершини, називається *простим ланцюгом.* Замкнутий ланцюг називається *циклом*, а простий замкнутий ланцюг – *простим циклом*.

Орієнтовані маршрути на орграфі визначаються аналогічно з тією різницею, що початкова вершина кожної подальшої дуги маршруту повинна співпадати з кінцевою вершиною попередньої дуги. Інакше кажучи, рух маршрутом допускається тільки в напрямах, вказаних стрілками.

Шлях – це маршрут, що не містить дуг, що повторюються. *Простий шлях* – це шлях, що не містить вершин, які повторюються (за виключенням, можливо, початку і кінця маршруту). Замкнутий шлях називається *контуром*, а простий замкнутий шлях – *простим контуром*. Граф (орграф) називається *циклічним (контурним),* якщо він містить хоч би один цикл (контур), інакше він називається *ациклічним (безконтурним)*.

Приклад 6. Розглянемо граф, показаний на рис. 1.2, *а*. На цьому графі: (e_1, e_3, e_2, e_3, e_5) – маршрут, що проходить через послідовність вершин (v_1, v_2, v_3, v_2, v_3, v_5) і сполучає вершини v_1 і v_5; (e_5, e_6, e_4, e_4) – маршрут, що проходить через послідовність вершин (v_3, v_5, v_5, v_2, v_5), сполучаючи v_3 і v_5; (e_1, e_3, e_5, e_4, e_1) – замкнутий маршрут; (e_2, e_5, e_6) – ланцюг; (e_1, e_2, e_5) – простий ланцюг; (e_2, e_3, e_4, e_5) – цикл; (e_2, e_4, e_5) – простий цикл.

На орграфі рис. 1.4, *а* маршрут (e_1, e_2, e_5) – простий шлях, що є контуром, а маршрут (e_1, e_2, e_3) – простий неконтурний шлях.

Цикл, який містить всі ребра графа, називається *ейлеровим циклом*, а граф, у якому є такий цикл, називається *ейлеровим графом*. Простий цикл, який

проходить через всі вершини графа, називають *гамільтоновим*. Якщо критерій існування ейлерового циклу дуже простий (необхідно, щоб степені всіх вершин були парними), то для гамільтонових циклів ніякого загального правила не знайдено.

Граф $G' = (V', E')$ є *частиною графа* $G = (V, V)$, якщо $V' \subset V$ і $E' \subset E$, тобто граф містить всі вершини і ребра будь-якої його частини.

Визначення. Підграф – це частина графа, яка, разом з деякою підмножиною ребер графа, містить і всі інцидентні ним вершини. Суграф – це частина графа, яка разом з деякою підмножиною ребер графа містить всі вершини графа $(V' = V, E' \subset E)$.

Початковий граф по відношенню до його підграфа називають *надграфом*, а по відношенню до суграфа – *понадграфом*. Сукупність всіх ребер графа, що не належать його підграфу (разом з інцидентними вершинами), утворює *доповнення підграфа*. Говорять, що підграфи одного графа $G' = (V', E)$ і $G'' = (V'', E'')$ *розділені ребрами*, якщо вони не мають загальних ребер $(E' \cap E'' = \varnothing)$ і *розділені вершинами*, якщо у них немає загальних вершин $(V' \cap V'' = \varnothing)$.

Приклад 7. На рис. 1.6 показані різні частини кубічного орграфа G.

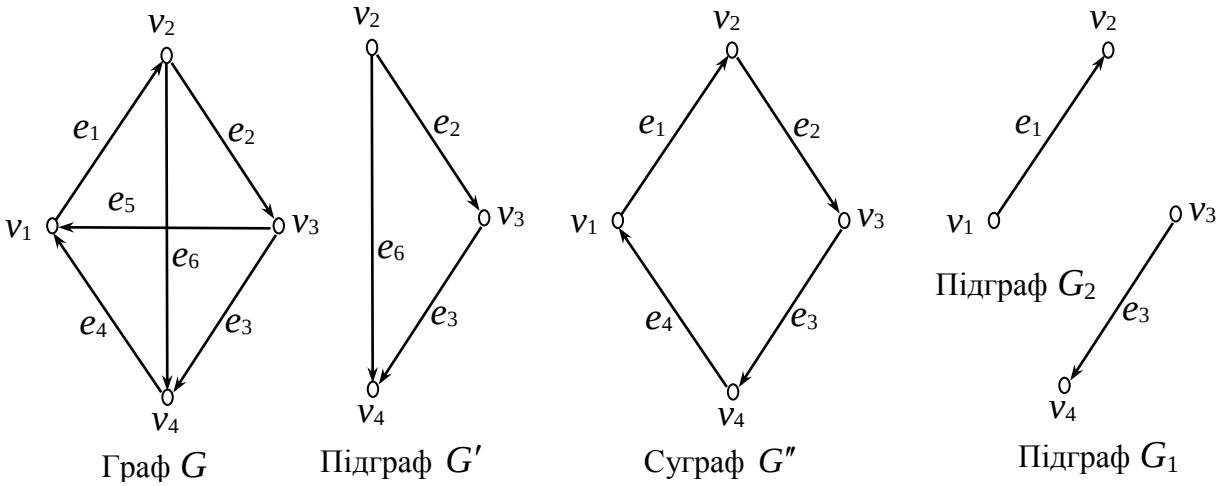


Рисунок 1.6 – Частини графа G

Різні частини кубічного орграфа G такі: підграфи G', G_1 та G_2; суграф G''. Підграфи G_1 та G_2 розділені ребрами і вершинами.

1.6. Зв'язність і роздільність

Визначення. Дві вершини графа називають зв'язаними, якщо існує маршрут, що сполучає ці вершини. Граф, будь-яка пара вершин якого зв'язана, називають зв'язним графом.

У зв'язному графі між будь-якими двома вершинами існує простий ланцюг, оскільки з маршруту, що зв'язує їх, завжди можна видалити циклічну ділянку, що проходить через деяку вершину більше одного разу.

Якщо граф незв'язний, то множину його вершин можна єдиним чином розділити на неперетинні підмножини, кожна з яких містить всі зв'язані між собою вершини і разом з інцидентними ним ребрами утворює зв'язний підграф. Таким чином, не зв'язний граф є сукупністю окремих частин (підграфів), які називають *компонентами*.

Часто відношення зв'язності ускладнюється додатковими умовами. Граф називають *циклічно зв'язним*, якщо будь-які дві різні вершини містяться в циклі. Граф називають *k-зв'язним*, якщо будь-яка пара різних вершин зв'язана, принаймні k ланцюгами, які не мають загальних вершин (крім початкової і кінцевої).

Зв'язність орієнтованих графів визначається так само, як і для неорієнтованих (без урахування напрямів дуг). Специфічним для орграфа (або змішаного графа) є поняття сильної зв'язності. Орграф називають *сильно зв'язним*, якщо для будь-якої пари його вершин v_i, v_j існує шлях з v_i в v_j та з v_j в v_i.

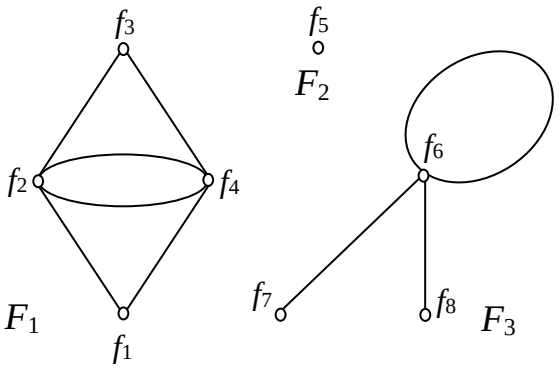


Рисунок. 1.7 – Незв'язний граф F

Приклад 8. На рис. 1.3, *б* показаний зв'язний граф. Граф F на рис. 1.7 – незв'язний. Він складається з трьох компонент F_1, F_2, F_3 (ізольована вершина також вважається компонентою). Компонента F_1 графа F циклічно зв'язна (однозв'язна), а зв'язана компонента F_3 не є циклічно зв'язаною, оскільки вершини f_7 і f_8 не містяться ні в якому циклі з іншими вершинами.

Орграф G на рис. 1.6 – сильно зв'язний. Також сильно зв'язним завжди повинен бути граф, що представляє план міста з одностороннім рухом деякими вулицями, оскільки інакше знайшлися б вершини (площі і перехрестя), між якими не можна було б проїхати містом без порушення правил руху.

Зв'язний граф може бути розділений на не зв'язні підграфи видаленням з нього деяких вершин і ребер (при видаленні вершин виключаються і всі інцидентні ним ребра, а при видаленні ребер вершини зберігаються).

Визначення. Якщо існує така вершина, видалення якої перетворює зв'язний граф (або компоненту незв'язного графа) у незв'язний, то вона називається точкою зчленування. Ребро з такими ж властивостями називається мостом.

Зрозуміло, що за наявності моста в графі є, принаймні, дві точки зчленування.

Граф називається *нероздільним*, якщо він зв'язний і не має точок зчленування. Граф, що має хоч би одну точку зчленування, є роздільним і називається *сепарабельним*. Він розбивається на блоки, кожен з яких є максимальним нероздільним підграфом. Кожне ребро графа, як і кожна вершина (за винятком точок зчленування), належать тільки одному з його блоків. Більш того, тільки одному блоку належить і кожен простий цикл.

Звідси випливає, що сукупністю блоків графа є розбиття множини ребер і простих циклів на неперетинні підмножини.

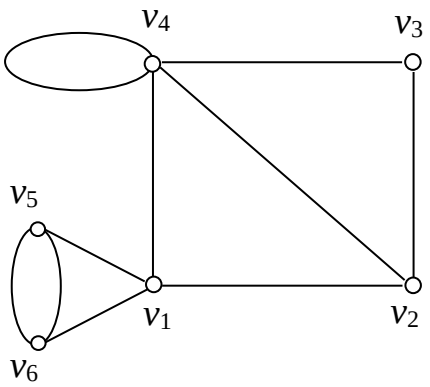


Рисунок 1.8 – Роздільний граф

Приклад 9. Граф G на рис. 1.6 неподільний. Зв'язний граф, представлений на рис. 1.8, має дві точки зчленування – v_1 і v_4, проте його ребро (v_1, v_4) не є міст. Граф B на рис. 1.9 має дві точки зчленування – v_4 і v_5, причому ребро (v_4, v_5), що з'єднує ці точки, є мостом, який розбиває даний граф на три блоки (блоки B_1, B_2 і B_3 на рис. 1.9). Кожен з цих блоків є нероздільним підграфом.

У ряді застосувань теорії графів блоки можна розглядати як компоненти. Це звичайно допустимо, коли зв'язки блоків за допомогою точки зчленування неістотні або коли істотні властивості графа пов'язані тільки з його простими циклами (контурами). У таких випадках можна розглядати незв'язний граф як зв'язний роздільний граф, який

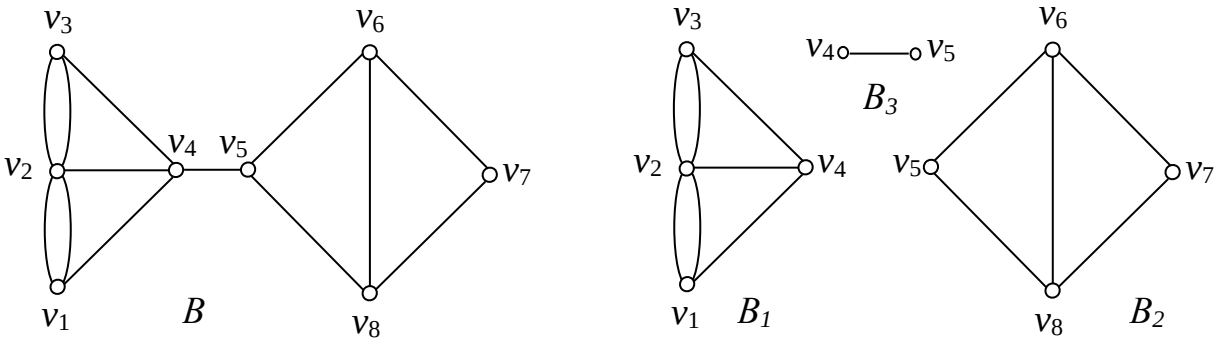


Рисунок 1.9 – Блоки роздільного графа B з мостом

утворюється шляхом такого об'єднання компонент, щоб кожна з них була блоком (це завжди можна зробити, об'єднавши, наприклад, по одній вершині кожного блока в точку зчленування).

1.7. Характеристики графів

Йдеться про числові характеристики графів.

Цикломатичне число графа G, який має n вершин, m ребер, r ділянок зв'язності, можна вирахувати за формулою (1.1)

$$v(G) = m - n + r. \qquad (1.1)$$

Число v є кількістю незалежних циклів на графі. Залежний цикл це той, що побудований тільки з ребер, які належать іншим циклам. В іншому випадку цикл є незалежним.

Хроматичне число графа *G*. Йдеться про таке “розфарбовування” вершин графа, коли суміжні вершини мали б різні кольори. Найменша кількість кольорів *p* має назву – хроматичне число графа, а граф є *p*-хроматичним, а також має позначення $\gamma(G)$, тобто $\gamma(G) = p$. Якщо $(G) = 2$, то граф є **біхроматичним.**

Існує множина евристичних процедур розфарбовування графів. Одним з методів є послідовний метод, оснований на впорядковуванні множини вершин. Спочатку вершини розташовуються в порядку незростання їх величин. Перша вершина зафарбовується в колір 1; потім список вершин проглядається зверху вниз (за незростанням степенів) і в колір 1 зафарбовується всяка вершина, яка не суміжна з іншою, вже пофарбованою в цей колір. Потім повертаємося до першої в списку нефарбованої вершини, зафарбовуємо її в колір 2 і знову проглядаємо список вершин зверху вниз, зафарбовуючи в колір 2 будь-яку нефарбовану вершину, яка не сполучена ребром з іншою, вже зафарбованою в колір 2 вершиною. Аналогічно діємо з фарбами 3, 4 і так далі, поки не будуть пофарбовані всі вершини. Число використаних фарб буде тоді наближеним значенням хроматичного числа графа.

Приклад 10. Розфарбувати граф *G* (рис. 1.10).

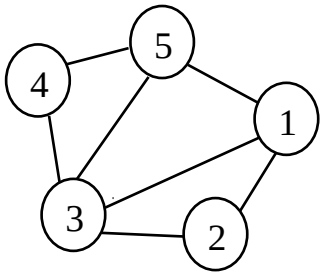
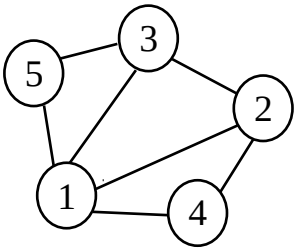


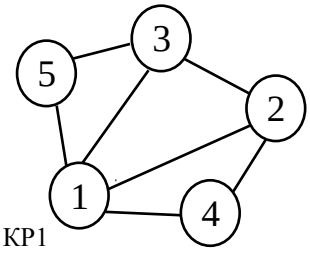
Рисунок 1.10 – Граф *G*

Алгоритм рішення.

Крок 1: вершини розташовуються в порядку зменшення їх степенів.

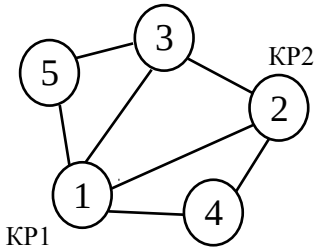


Крок 2: нефарбована вершина з найменшим номером зафарбовується в колір 1, потім вершини розташовуються в порядку зростання номерів. Для графа *G* нефарбована вершина з найменшим номером – вершина 1. Розфарбовуємо цю вершину в колір номер 1 (КР1).

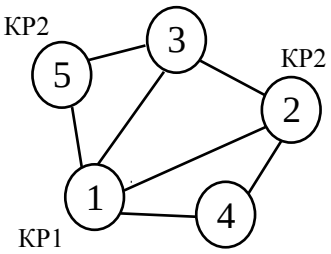


КР1

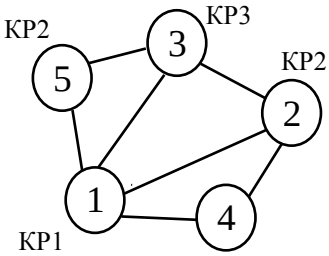
Крок 3: вершини 2, 3, 4, 5, суміжні з однією із зафарбованих в колір 1, не можуть бути розфарбовані в цей колір. Вибираємо нефарбовану вершину з найменшим номером. Це вершина 2. Розфарбовуємо цю вершину в колір номер 2 (КР2).



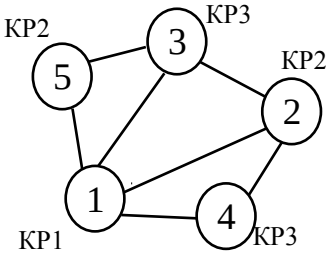
Крок 4: оскільки 3 і 4 вершини суміжні з вершиною із меншим номером, що має колір номер 2, вони не можуть бути розфарбовані цим кольором. Вершина 5 не суміжна ні з однією вершиною із меншим номером, що має колір номер 2. Розфарбовуємо цю вершину в колір номер 2 (КР2).



Крок 5: вибираємо наступну нефарбовану вершину з найменшим номером. Це вершина 3. Розфарбовуємо цю вершину в колір номер 3 (КР3).



Крок 6: оскільки 4 вершина не суміжна ні з однією вершиною із меншим номером, що має колір номер 3, розфарбовуємо її в колір номер 3 (КР3).



1.8. Дерева і ліс

Нехай множина V містить p вершин, які пронумеровані порядковими числами від 1 до p, тобто $V = \{1, 2, ..., p\}$. *Зв'язавши ці вершини ребрами так, щоб були відсутні цикли, отримаємо деякий граф, що покриває дану множину p вершин.*

Визначення. Дерево – це зв'язний ациклічний граф.

Дерево на множині p вершин завжди містить $q = p - 1$ ребро, тобто мінімальну кількість ребер, необхідну для того, щоб граф був зв'язним. Дійсно, дві вершини зв'язуються одним ребром, і для зв'язку кожної подальшої вершини з попередніми потрібне ребро, отже, для зв'язку p вершин необхідно і достатньо $p - 1$ ребро. При $p = 2$ дерево одне і воно складається з однієї гілки.

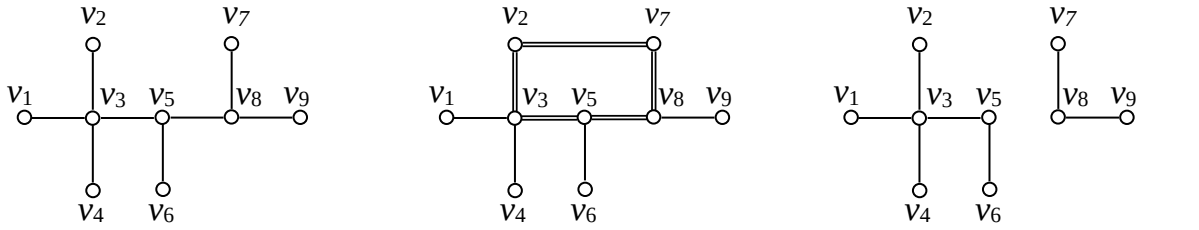
При додаванні в дерево ребра утворюється цикл, а при видаленні хоч би одного ребра дерево розпадається на компоненти, кожна з яких є також деревом або ізольованою вершиною.

Визначення. Незв'язний граф, компоненти якого є деревами, називається лісом.

Ліс з k дерев, що містить p вершин, має в точності $p - k$ ребер. Дерева вважаються *істотно різними*, якщо вони не ізоморфні. Зі збільшенням числа вершин кількість різних дерев різко зростає (наприклад, при $p = 20$ їх налічується близько мільйона). Серед різних дерев виділяються два важливі окремі випадки: *послідовне дерево,* що є простим ланцюгом, і *зіркове дерево*, в якому одна з вершин (центр) суміжна з рештою всіх вершин.

Розглядаються також дерева з орієнтованими ребрами (дугами). Орієнтоване дерево називається *прадеревом* з коренем v_0, якщо існує шлях між вершиною v_0 і будь-якою іншою його вершиною. Ясно, що прадерево має єдиний корінь (вершину, в яку не входить жодна дуга).

Приклад 11. Дерево T на рис. 1.11 має 9 вершин і 8 ребер. Введення додаткового ребра (v_2, v_7) порушує ациклічність дерева і перетворює його на зв'язний граф T_1, в якому цикл (v_2, v_7, v_8, v_5, v_3, v_2) виділений подвійною лінією. Видалення ребра (v_5, v_8) перетворює дерево T в ліс, що складається з двох дерев, – T_2 і T_3.



Дерево T Граф T_1 Дерево T_2 Дерево T_3

Рисунок 1.11 – Дерево T і варіанти його зміни

На рис. 1.12 показані всі істотно різні шестивершинні дерева ($T_1 - T_6$), серед яких є послідовне дерево – T_1 і зіркове дерево – T_6.

На рис. 1.13 показане дерево з орієнтованими ребрами, яке, очевидно, є прадеревом з коренем v_0.

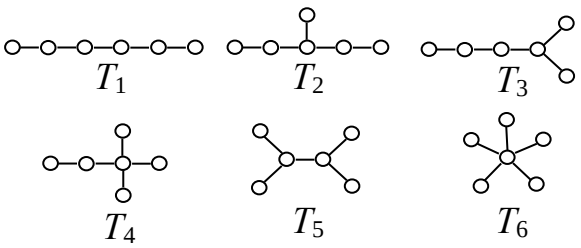


Рисунок 1.12 – Шестивершинні дерева

До цих пір розглядалися дерева як мінімальні зв'язні графи на множині p вершин. Важливе значення має й інша точка зору, коли дерева або ліс є частками деякого графа, тобто утворюються з його ребер. Будь-яка зв'язна сукупність ребер, що не містить контурів, разом з інцидентними ним вершинами утворює *дерево графа*. Якщо таке дерево є суграфом (містить всі вершини графа), то воно називається *покриваючим деревом* або *остовом*. Оскільки петля є простим циклом, що

складається з єдиного ребра, то вона не може входити до складу будь-якого дерева графа.

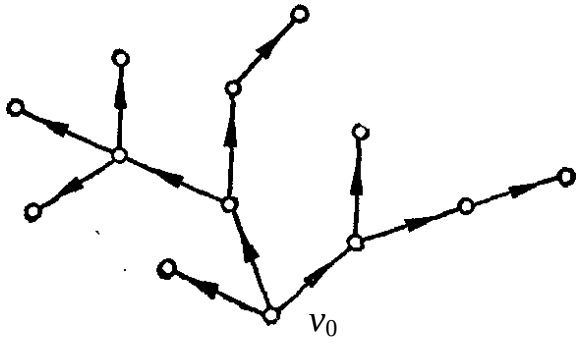


Рисунок 1.13 – Прадерево з коренем

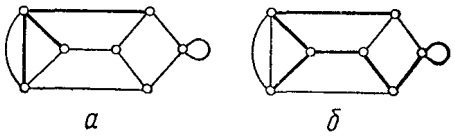


Рисунок 1.14 – Дерево як частина графа (виділено жирними лініями): *а* – дерево; *б* – остов (покривне дерево)

Ребра графа, які належать його дереву, називають *гілками*. Якщо дерево покриває граф, то множина ребер графа розбивається на дві підмножини: підмножина гілок і підмножина ребер *доповнення ребер*, названих *хордами*. При цьому зв'язний *(p, q)-граф містить* $\nu = p - 1$ гілок і $\sigma = q - p + 1$ хорд. Якщо граф незв'язний, то сукупність остовів k його компонент утворює *покривний* ліс. У цьому випадку $\nu = p - k$ і $\sigma = q - p + k$.

Дерева мають важливе значення в різних прикладних задачах, коли, наприклад, мова йде про з'єднання яких-небудь об'єктів мінімальним числом каналів (ліній зв'язку, доріг, комунікацій) з певними властивостями. За допомогою дерева визначається система координат при моделюванні ланцюгів і систем різної фізичної природи. Дерева використовуються як моделі при розгляді ієрархічних систем об'єктів, структурних формул органічних сполук і тому подібне.

1.9. Приклади задач, які використовують зважені графи

1.9.1. Аналіз електричних кіл

У комп'ютерних пакетах програм автоматичного аналізу електричних кіл алгоритми на графах є неодмінним засобом.

Під час створення схеми електричного кола на екрані програмна система будує зважений граф кола, а потім – сигнальний граф. Шляхом спрощень сигнального графа можна одержати коефіцієнт передачі кола та інші характеристики.

1.9.2. Задача про найкоротший шлях

Зважений граф має вагу кожного ребра, яка відповідає довжині ребра. Потрібно відшукати найкоротший шлях (найменшу суму ваг ребер) між вершиною входу та вершиною виходу графа.

Задача полягає в знаходженні зв'язаних між собою шляхів на транспортній мережі, які в сукупності мають мінімальну довжину від початкового пункту до пункту призначення. Постановка задачі набуває смислу в тому випадку, якщо є декілька варіантів маршруту з початкового пункту в

кінцевий. В цьому випадку зміст задачі полягає в мінімізації загальної довжини маршруту.

Введемо позначення:

d_{ij} – відстань на мережі між суміжними вузлами i та j;

U_j – найкоротша відстань між вузлами i та j, $U_1 = 0$.

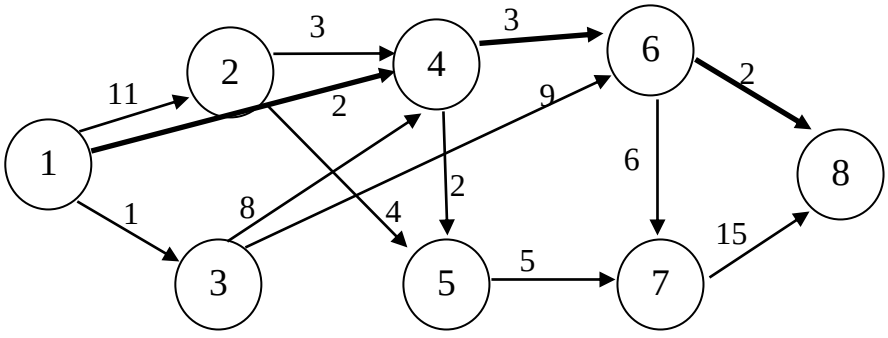
Формула для обчислення U_j (1.2):

$$U_j = \min_i \left\{ \begin{array}{l} \text{Найкоротша відстань до попереднього} \\ \text{вузла } i \text{ плюс відстань між поточним} \\ \text{вузлом } j \text{ і попереднім вузлом } i \end{array} \right\} = \min_i \{U_i + d_{ij}\} \quad (1.2)$$

З формули випливає, що найкоротшу відстань U_j до вузла j можна обчислити лише після того, як визначена найкоротша відстань до кожного попереднього вузла i, сполученого дугою з вузлом j. Процедура завершується, коли отримано U_j останньої ланки.

Приклад 12. Знайти найкоротшу відстань між вузлами 1 та 8, якщо відстані між проміжними пунктами надані у такій таблиці (графічно зобразити відповідну мережу та стрілками відмітити напрями).

	1	2	3	4	5	6	7	8
1		11	1	2				
2				3	4			
3				8		9		
4					2	3		
5							5	
6							6	2
7								15



1) $U1 = 0$;

2) $U2 = \{ U1 + 11\} = 11$;

3) $U3 = \{ U1 + 1\} = 1$;

4) $U4 = \min\{ \underline{U1 + 2}, U2 + 3, U3 + 8\} = \min\{ 2, 14, 8\} = 2$;

5) $U5 = \min\{ U2 + 4, U4 + 2\} = \min\{ 15, 4\} = 4$;
6) $U6 = \min\{ U4 + 3, U3 + 9\} = \min\{ 5, 10\} = 5$;
7) $U7 = \min\{ U5 + 5, U6 + 6\} = \min\{ 9, 11\} = 9$;
8) $U8 = \min\{ U6 + 2, U7 + 15\} = \min\{ 7, 24\} = 7$.

Мінімальна відстань між вузлами 1 і 8 дорівнює 7, а відповідний маршрут: $U1 \rightarrow U4 \rightarrow U6 \rightarrow U8$.

1.9.3. Задача побудови графа найменшої довжини (завдання мінімізації мережі)

Задача мінімізації мережі полягає в знаходженні ребер, що сполучають всі вузли мережі та мають мінімальну сумарну довжину.

Така задача виникає при плануванні доріг, трубопроводів, ліній електропередач, ліній електрозв'язку та таке інше.

На площині розташовані вершини графа, їх координати відомі. Треба з'єднати вершини ребрами так, щоб загальна довжина всіх ребер була найменшою. Алгоритм Краскала полягає у такому. Обчислюють відстані між всіма парами вершин та шукають найменшу відстань. Відповідні вершини з'єднують ребром та відмічають як з'єднані. Перебирають всі з'єднані вершини і шукають серед нез'єднаних вершин найближчу до кожної, та вибирають пару з'єднана-нез'єднана з найменшою відстанню для з'єднання ребром. Операції продовжують поки всі вершини будуть з'єднані. Результат – це дерево найменшої довжини. Рішення є однозначним за умови різних відстаней між вершинами.

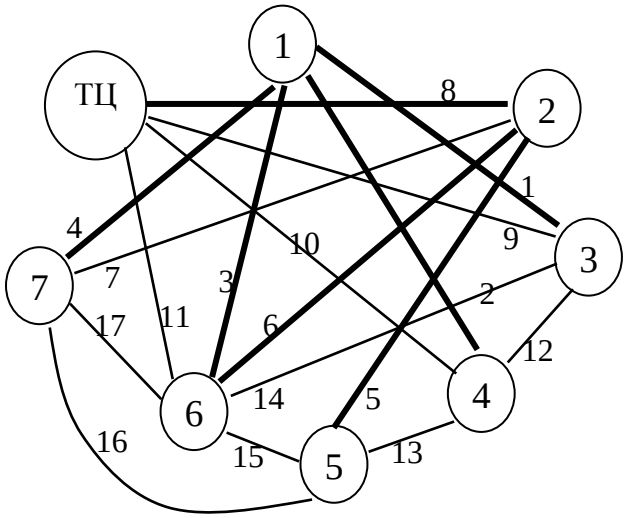
Приклад 13. Знайти мінімально необхідну довжину кабелю, який треба прокласти для встановлення зв’язку 7-ми мікрорайонів міста з центром прийому кабельного телебачення (відстані задані у км).

	ТЦ	1	2	3	4	5	6	7
ТЦ	-		8	9	10		11	
1		-		1	2		3	4
2			-			5	6	7
3				-	12		14	
4					-	13		
5						-	15	16
6								17
7								-

Алгоритм рішення.

Почнемо з будь-якого вузла і з'єднаємо його з найближчим вузлом мережі. Сполучені два вузли утворюють зв'язну множину, а решта – незв'язну. Далі в незв'язній множині виберемо вузол, який розташований ближче за інших до будь-якого з вузлів зв'язної множини. Скоректуємо

зв'язну і незв'язну множини і повторюватимемо процес до тих пір, поки в зв'язну множину не потраплять всі вузли мережі.



У разі однаково віддалених вузлів виберемо будь-який з них, що указує на неоднозначність (альтернативність) «мінімального дерева – остова».

1) {ТЦ} $S = 0$;
2) {ТЦ, 2} $S = 0 + 8 = 8$;
3) {ТЦ, 2, 5} $S = 8 + 5 = 13$;
4) {ТЦ, 2, 5, 6} $S = 13 + 6 = 19$;
5) {ТЦ, 2, 5, 6, 1} $S = 19 + 3 = 22$;
6) {ТЦ, 2, 5, 6, 1, 3} $S = 22 + 1 = 23$;
7) {ТЦ, 2, 5, 6, 1, 3, 4} $S = 23 + 2 = 25$;
8) {ТЦ, 2, 5, 6, 1, 3, 4, 1} $S = 25 + 4 = 29$.

Відповідь: 29 км.

1.9.4. Задача комівояжера

Комівояжер (франц.) – роз'їзний представник торговельної фірми, що пропонує покупцям товари за наявними у нього зразками. Маючи перелік адрес для візитів, він змушений завжди ретельно складати план – послідовність візитів, інакше зростуть транспортні видатки. Якщо відоме розташування місць відвідування – вершин графа, то спрощена задача є задачею побудови гамільтонового циклу мінімальної довжини. У загальному випадку, починаючи зі стартової вершини, комівояжер має $n-1$ напрямків руху. Вибравши вершину, комівояжер з неї має $n-2$ напрямків руху. Очевидно, що кількість гамільтонових циклів дорівнюватиме $(n-1)!$. Таке ціле число, якщо n перевищує 15, не можна розмістити в межах розрядної сітки сучасного комп'ютера і рішення задачі простим перебиранням шляхів з обчисленням їх довжини та пошуком мінімального за довжиною шляху може тривати дуже довго. Тобто – з таким алгоритмом ця задача не для комп'ютера і, взагалі, невідомо для яких засобів. Ефективного алгоритму для загального випадку досі не запропоновано. Є алгоритми для деяких випадків з обмеженнями, але вони необов'язково гарантують оптимальне рішення.

1.9.5. Мережне планування (PERT-Performance Evaluations and Review Techniques)

Це планування за допомогою мережевих моделей або інформаційних моделей комплексу взаємопов'язаних робіт. Така модель відображує часткову впорядкованість робіт у часі (можливі додаткові показники та параметри). Яка потреба у такому плануванні? Справа у тому, що швидке виконання науково-технічних розробок або будівництва великого обсягу неможливе без розпаралелювання роботи, залучення багатьох колективів виконавців та техніки. Без чіткого плану з визначенням строків виконання окремих робіт неможливо запобігти величезним втратам часу та матеріальних цінностей. Найбільш поширене подання мережевого плану у вигляді мережевого графіка. Це орграф без контурів, де вершини та дуги відтворюють відношення черговості між роботами. Дугами графа можуть бути роботи, вага дуг – тривалість виконання робіт. Вершини – це відображення подій з логічним зв'язком – кон'юнкцією відносно робіт-дуг, які входять у вершину. Тобто, роботи, позначені як дуги, що виходять з вершини, можуть бути розпочаті лише за умови виконання всіх робіт, що позначені дугами, які входять у вершину. Шлях на такому орграфі з початкової вершини у кінцеву, який має найбільшу тривалість у часі, має назву *критичний*. Він дає можливість вирахувати мінімальний час виконання робіт та побачити резерви, пов'язані з неповним завантаженням окремих виконавців.

1.9.6. Мережі Петрі

Це графічна модель системи з високим ступенем розпаралелювання обчислень (робіт). Вона має вершини різних типів і дуги транспортного та аналітичного типів та використовується для аналізу деяких властивостей систем.

1.9.7. Транспортні мережі та задача про найбільший потік

Транспортна мережа – скінченний орграф без петель, у якого

1) є вхід – вершина *s*;

2) є вихід – вершина *t*;

3) кожна дуга *u* має вагу *c*, яка означає пропускну здатність дуги, *c* – ціле число.

Потік у дузі це функція $\varphi(u)$, $0 \le u \le c(u)$, $u \in U$, U – множина дуг.

Дугу *u* називають насиченою, якщо $\varphi(u) = c(u)$, тобто, потік відповідає пропускній здатності. Алгоритм Форда-Фолкерсона знаходження повного та найбільшого потоку полягає у наступному:

1) знаходять ненасичений шлях з вершини *s* у вершину *t*;

2) нарощують потік, кожен раз перевіряючи, чи не стала насиченою яка-небудь дуга на шляху, якщо неможливо виконати пункт 1, тоді переходимо до пункту 3;

3) якщо на будь-якому шляху з вершини *s* у вершину *t* є хоча б одна насичена дуга – сумарний потік з вершини *s* або у вершину *t* є повним.

Алгоритм знаходження найбільшого потоку полягає у такому: якщо можливо, що інверсія напрямку потоку у деяких дугах підвищує повний потік,

шукають найбільший повний потік за умови використання можливих змін напрямків потоку в окремих дугах. Такий потік є найбільшим потоком мережі.

Якщо, крім пропускної здатності дуг, відома і вартість транспортування одиниці потоку, то одержуємо задачу раціонального розподілу потоку у дугах з метою зменшення витрат. Це транспортна задача – задача найекономнішого розподілу потоку у дугах. Рішенням є раціональний план транспортування.

Контрольні запитання

1. Що називається графом?
2. Що таке оргграф?
3. Якщо у графа не всі ребра орієнтовані, то цей граф:
   A) однорідний;
   B) змішаний;
   C) орієнтований;
   D) частково-орієнтований.
4. Висяча вершина – це:
   A) вершина, яка не має інцидентних ребер;
   B) вершина, яка інцидентна єдиному ребру;
   C) спеціально виділена вершина дерева;
   D) вершина, якій інцидентні тільки дуги, що заходять у вершину.
5. Мультиграф – це:
   A) зв'язний ациклічний граф;
   B) граф, який має кратні (паралельні) ребра;
   C) планарний граф, що укладений на площину;
   D) зв'язний граф, що не містить вершин непарного ступеня.
6. Повний граф на *N* вершинах – це:
   A) однорідний граф $N + 1$степеня;
   B) однорідний граф $N - 1$ степеня;
   C) однорідний розріз;
   D) планарний граф.
7. Граф з петлями та кратними ребрами називається:
   A) плоским;
   B) зв’язним;
   C) псевдографом;
   D) простим або звичайним.
8. Граф, у якого ступені усіх вершин однакові, називається:
   A) повним;
   B) однорідним (регулярним);
   C) інцидентним;
   D) парним.

9. Граф, що містить петлі, називається:

A) псевдографом;

B) планарним графом;

C) нуль-графом;

D) плоским графом.

10. Які із вказаних ребер у даному графі є петлями:

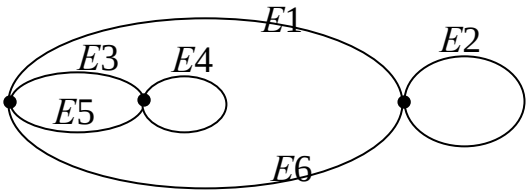
A) *E*1, *E*6;

B) *E*4, *E*2;

C) *E*3, *E*5;

D) *E*3, *E*5, *E*1, *E*6;

E) *E*1, *E*6, *E*2?



11. Дві вершини називаються суміжними, якщо...

A) вони є межевими вершинами ланцюга;

B) вони є межевими вершинами маршруту;

C) вони є межевими вершинами одного ребра;

D) вони входять в один цикл.

12. Степінь (валентність) вершини – це:

A) число інцидентних їй ребер;

B) число інцедентних ребер плюс число суміжних вершин;

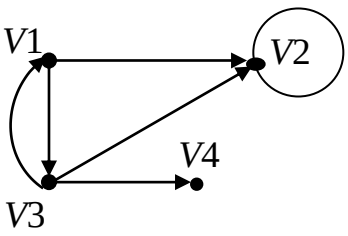
C) число інцидентних ребер плюс число інцедентних петель;

D) число інцедентних дуг, що заходять до вершин.

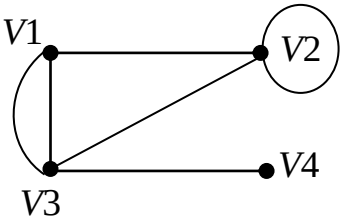
13. Задана матриця суміжності. Якому з наведених графів вона відповідає?

	*V*1	*V*2	*V*3	*V*4
*V*1	0	1	1	0
*V*2	0	1	0	0
*V*3	1	1	0	1
*V*4	0	0	0	0

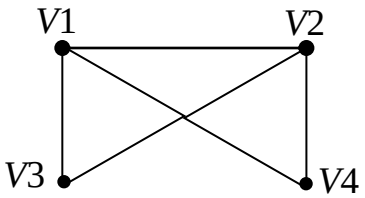
1



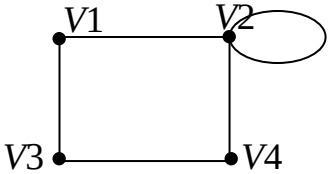
2



3



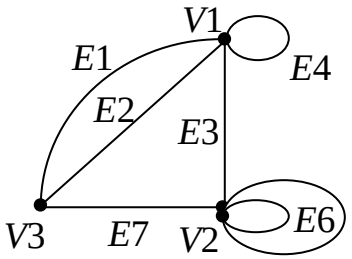
4



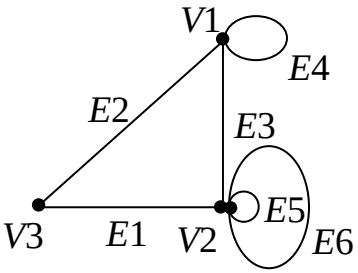
14. Якому графу відповідає дана матриця суміжності?

	*V*1	*V*2	*V*3
*V*1	1	1	2
*V*2	1	2	1
*V*3	2	1	0

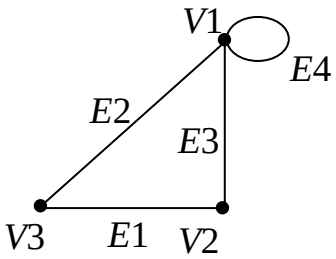
1



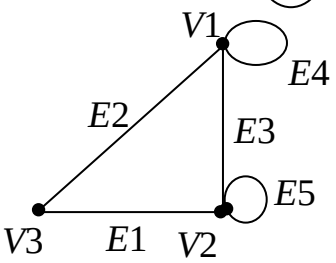
2



3



4



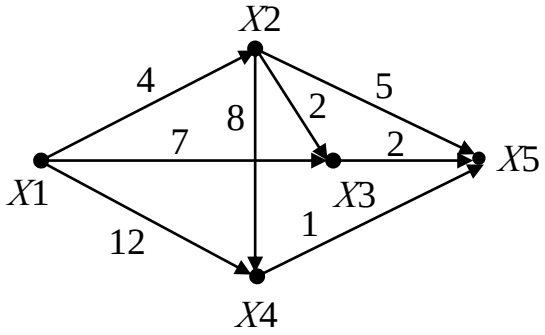
15. Вказати найкоротший шлях з вершини *X*1 у *X*5 в даному графі

A) *X*1, *X*2, *X*5;

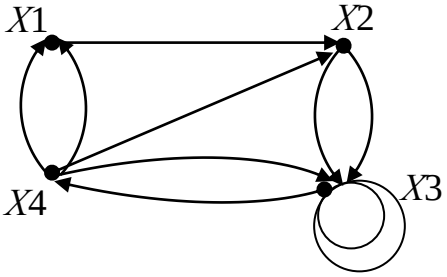
B) *X*1, *X*4, *X*5;

C) *X*1, *X*2, *X*3, *X*5;

D) *X*1, *X*3, *X*5.



16. Якій матриці суміжності відповідає даний граф?



1

	*X*1	*X*2	*X*3	*X*4
*X*1	0	1	0	0
*X*2	0	0	2	0
*X*3	0	0	2	1
*X*4	2	1	1	0

2

	*X*1	*X*2	*X*3	*X*4
*X*1	0	1	0	1
*X*2	0	0	1	0
*X*3	0	1	2	1
*X*4	1	1	1	0

3

	*X*1	*X*2	*X*3	*X*4
*X*1	2	1	0	1
*X*2	0	1	1	0
*X*3	0	1	0	1
*X*4	1	1	1	0

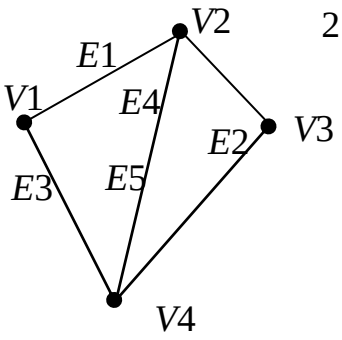
4

	*X*1	*X*2	*X*3	*X*4
*X*1	1	0	0	2
*X*2	0	1	1	0
*X*3	0	1	0	1
*X*4	1	1	1	0

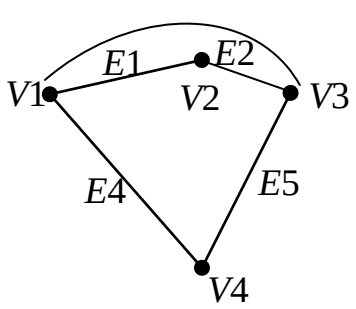
17. Задана матриця інцидентності неорієнтованого графа. Якому з наведених графів вона відповідає?

	*E*1	*E*2	*E*3	*E*4	*E*5
*V*1	1	1	0	0	0
*V*2	1	1	1	0	0
*V*3	0	0	1	1	1
*V*4	0	0	0	1	1

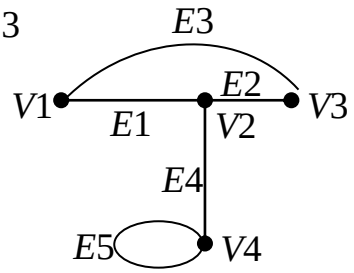
1



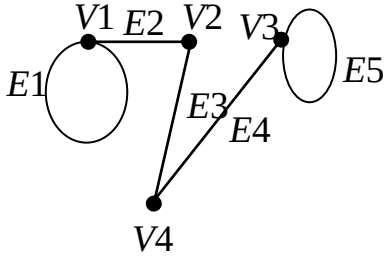
2



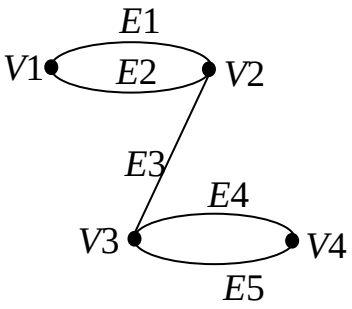
3



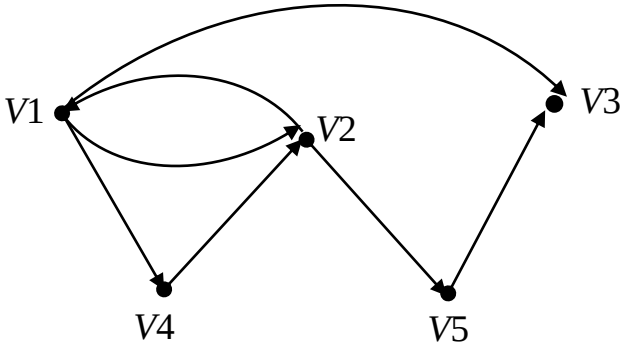
4



5



18. Яка з наведених матриць суміжності відповідає заданому графу?



1

	*V*1	*V*2	*V*3	*V*4	*V*5
*V*1	0	1	1	1	0
*V*2	1	0	0	0	1
*V*3	0	0	1	1	0
*V*4	0	1	0	0	0
*V*5	0	0	0	0	0

2

	*V*1	*V*2	*V*3	*V*4	*V*5
*V*1	0	1	1	1	0
*V*2	1	0	0	0	1
*V*3	0	0	0	0	0
*V*4	0	1	0	0	0
*V*5	0	0	1	0	0

3

	*V*1	*V*2	*V*3	*V*4	*V*5
*V*1	1	0	1	1	0
*V*2	1	0	0	0	1
*V*3	0	0	0	0	0
*V*4	0	1	0	0	0
*V*5	0	0	0	0	2

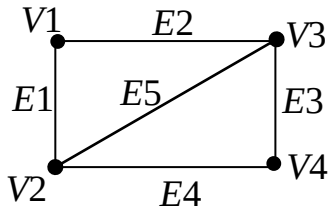
4

	*V*1	*V*2	*V*3	*V*4	*V*5
*V*1	0	0	1	0	1
*V*2	1	0	0	0	1
*V*3	0	0	0	0	0
*V*4	0	1	0	0	0
*V*5	0	1	1	0	0

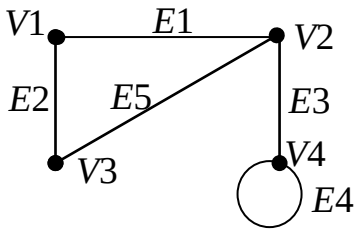
19. Задана матриця інцидентності неорієнтованого графа. Якому з наведених графів вона відповідає?

	*E*1	*E*2	*E*3	*E*4	*E*5
*V*1	1	1	0	1	0
*V*2	1	0	0	0	1
*V*3	0	1	1	0	0
*V*4	0	0	1	1	1

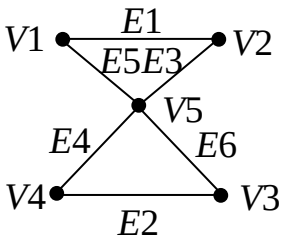
1



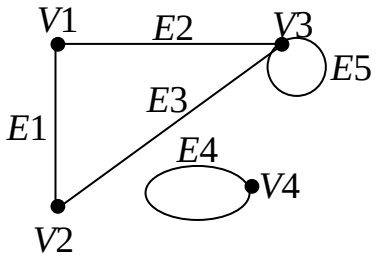
2



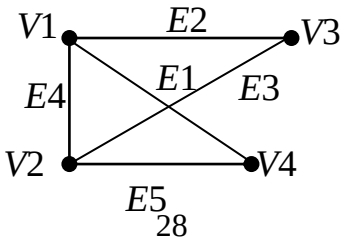
3



4



5



20. Простий ланцюг – це:

A) ланцюг, який не має найменшої величини;

B) ланцюг, в якому всі вершини відмінні;

C) ланцюг, який містить всі ребра графа тільки один раз;

D) ланцюг з мінімальним числом ребер.

21. Маршрут – це:

A) простий ланцюг, у якого усі вершини різні;

B) переміжна послідовність вершин та ребер, яка має таку властивість, що будь-яка пара сусідніх елементів інцидентна;

C) шлях, у якого початкова і кінцева вершини співпадають;

D) ланцюг, що містить усі ребра графа рівно один раз.

22. Як називається ланцюг, вершини якого не повторюються:

A) простий ланцюг;

B) звичайний ланцюг;

C) шлях;

D) ейлерів ланцюг.

23. Гамільтоновим циклом у графі називається цикл, ...

A) що містить усі ребра графа G рівно один раз;

B) що містить $(n-1)$ ребро графа G рівно один раз;

C) усі ребра якого різняться;

D) що містить усі вершини графа G рівно один раз.

24. Точка зчленування – це:

A) вершина, видалення якої з графа призведе до зменшення кількості компонент зв'язності;

B) вершина інцидентна одному ребру;

C) вершина, видалення якої з графа збільшує число компонент зв'язності цього графа;

D) вершина, що зв'язує два ребра.

25. Міст – це:

A) ребро, що інцидентне висячій вершині;

B) вершина, видалення якої з графа збільшує кількість компонент зв'язності;

C) ребро, що з'єднує дві вершини;

D) ребро графа, видалення якого зменшує кількість його компонент зв'язності.

26. Цикломатичне число графа дорівнює (де m – кількість ребер у графі, n – кількість вершин у графі, k – число компонент зв’язності графа):

A) $v = n - m - k$;

B) $v = n - m + k$;

C) $v = 2m - n + k$;

D) $v = m - n + 2k$.

27. Яку кількість ребер містить дерево з N вершинами:

A) $N - 2$;

B) $N - 1$;

C) N;

D) $N + 1$;

E) $N + 2$?

28. Як називається зв'язний граф, що не містить циклів:

A) ліс;

B) дерево;

C) болото;

D) ейлерів?

29. Задано матрицю суміжності орграфа. Побудувати матрицю інцидентності, матрицю досяжності, список дуг.

1

	*v*1	*v*2	*v*3	*v*4	*v*5
*v*1	1	1	0	0	0
*v*2	0	1	0	0	1
*v*3	0	1	0	1	0
*v*4	0	1	0	1	0
*v*5	0	0	1	0	0

2

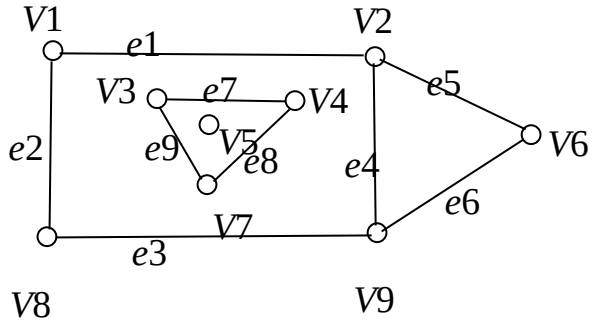
	*v*1	*v*2	*v*3	*v*4	*v*5
*v*1	1	0	0	1	1
*v*2	0	1	0	1	1
*v*3	0	0	1	0	0
*v*4	1	1	0	1	1
*v*5	1	1	0	1	1

3

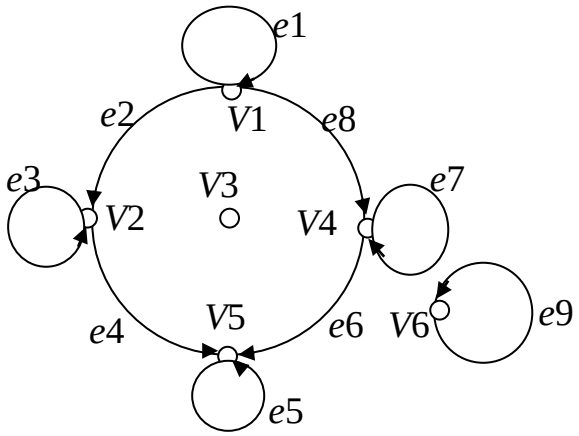
	*v*1	*v*2	*v*3	*v*4	*v*5	*v*6
*v*1	0	1	0	0	0	1
*v*2	0	0	0	1	0	0
*v*3	1	0	0	0	1	0
*v*4	1	0	1	0	0	1
*v*5	0	0	0	1	0	1
*v*6	0	1	0	0	0	1

30. Для графа *G* задати матрицю інцидентності, матрицю суміжності, матрицю досяжності. Визначити хроматичне та цикломатичне числа.

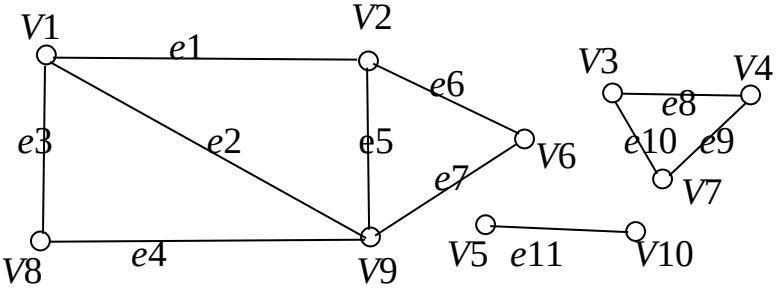
1



2



3



31. Знайти мінімально необхідну довжину кабелю, який треба прокласти для встановлення зв’язку 7-ми мікрорайонів міста з центром прийому кабельного телебачення (відстані задані у км).

	ТЦ	1	2	3	4	5	6	7
ТЦ	-	3	17	8	15	4		
1		-	16		13		5	
			-		2	9		6
3				-	14	12		5
4					-	10	4	1
5						-	2	12
6							-	7
7								-

32. Знайти найкоротшу відстань між вузлами 1 та 8, якщо відстані між проміжними пунктами надані у такій таблиці:

	1	2	3	4	5	6	7	8
1		3	7	8				
2				5	6			
3				4		7		
4					11	12		
5							8	
6							4	8
7								28

2. ЛОГІКА

Математична логіка – це формальна система, носієм якої є символи і послідовності символів формальної мови, а безліч операцій використовується для формування і виведення нових висновків формальної мови. Між символами немає інших зв'язків і відношень, окрім тих, які описані засобами формальної системи.

Математична логіка розглядає мови, основна мета яких – забезпечити символізм (систему формальних позначень) для міркувань, що зустрічаються не тільки в математиці, але і в повсякденному житті.

Виділяють декілька типів формальних систем: логіка висловів і логіка предикатів, логіка відношень і нечітка логіка та ін.

Логіка висловів (*prepositional calculus*) – це модель формальної системи, предметом якої є оповідні речення, взяті цілком без урахування їх внутрішньої структури.

Логіка предикатів (*predicate calculus*) – це формальна система, предметом якої є речення з урахуванням його внутрішнього складу і структури.

Логіка відношень (*relation calculus)* – це формальна система, предметом якої є відношення у вигляді множини однорідних речень, що істотно розширюють логіку предикатів. Частіше цю логіку називають реляційною.

Логіка нечітка (*fuzzi calculus)* – це формальна система, предметом якої є речення при нечіткому заданні характерних ознак окремих складових елементів, або відношень між ними.

Двозначна логіка має справу з такими об'єктами, які приймають одне з двох можливих значень (істинний або хибний вислів, висока або низька напруга, наявність або відсутність заданої ознаки в об'єкта і т. ін.). Об'єкти, які можуть набувати значень зі скінченної множини, що містить більше двох елементів, називають *багатозначними.* Вони або зводяться яким-небудь способом до двозначних об'єктів, або обслуговуються апаратом *багатозначної логіки***.**

Об'єктами математичної логіки є будь-які дискретні скінченні системи, а її **головне завдання –** структурне моделювання таких систем.

Об'єкти з двома можливими станами характеризуються *булевими змінними,* які здатні набувати лише два різні значення. Для позначення цих двох значень зазвичай використовуються цифри 0 та 1 або букви *х* (хибно) та *і* (істинно).

Відношення між булевими змінними представляються *булевими функціями,* які подібно до числових функцій можуть залежати від однієї, двох і, взагалі, *n* змінних (аргументів). Запис $y = f(x_1, x_2 ..., x_n)$ означає, що y – функція аргументів $x_1, x_2 ..., x_n$. Найважливіша особливість булевих функцій полягає в тому, що вони, як і їх аргументи, набувають своїх значень з двоелементної множини {0,1}, або (*і, х*), тобто характеризуються одним з двох можливих станів.

2.1. Логіка висловів. Загальні поняття

Первинним поняттям математичної логіки є «вислів». Будь-яке стверджувальне речення, яке може бути визнане істинним або хибним (але не тим та іншим одночасно), називають *висловом*. Приклади висловів: «Сніг білий», «Цукор – вуглеводень», «Місяць більше Землі». Такі вислови називають простими або *елементарними*.

При формальному дослідженні складних текстів поняття «Прості вислови» заміщають поняттям «*Пропозиційні змінні*», які позначають прописними буквами латинського алфавіту «*A*», «*B*», «*C*». «Істина» або «хибність», приписана деякому вислову, називається *істиннісним значенням* цього вислову. Істинність або хибність вислову відзначають символами «1» – *істина* або «0» – *хибність*.

Наприклад, можна позначити вислови таким чином:

P = Сніг білий;

Q = Цукор – вуглеводень;

R = Місяць більше Землі.

Символи *P, Q, R* і т. д., які використовуються для позначення висловів, називаються *атомарними формулами* або *атомами.*

Тоді істиннісне значення формул таке: *P* = 1; *Q* = 1; *R* = 0.

Якщо є декілька простих висловів, то за допомогою логічних зв'язок і заперечень з них можна утворити різні нові вислови, які називаються *складними*.

Приклад 1. Нехай є вислови: «На вулиці світить сонце», «У класі йдуть заняття». З цих простих висловів можливо різними способами побудувати складні вислови:

1) На вулиці світить сонце, і в класі йдуть заняття;

2) На вулиці не світить сонце;

3) На вулиці світить сонце, проте в класі йдуть заняття;

4) У класі йдуть заняття, а на вулиці світить сонце;

5) Або на вулиці світить сонце, або в класі йдуть заняття;

6) Якщо на вулиці світить сонце, то в класі йдуть заняття;

7) Якщо в класі йдуть заняття, то на вулиці світить сонце.

В алгебрі висловів допускаються будь-які граматично правильні способи утворення складних висловів й абсолютно ігнорується смислова характеристика отриманого речення.

Будь-яке з наведених складних речень допустимо з погляду алгебри висловів і називається *правильно побудованою формулою*.

Правила виконання логічних операцій над висловами формує *алгебру висловів*. Правила виведення нових висловів, основані на аксіомах і відношеннях між висловами, формують числення висловів. Вислови, з яких роблять виведення нових висловів, називають *засновком*, а отримуваний вислів – *висновком*.

Множина $V = \{A, B, C, ...\}$ із заданими над нею логічними операціями $F = \{\neg; \&; \vee; \rightarrow; \leftrightarrow\}$ формують *модель алгебри висловів* $A_в = <V; F>$.

У логіці висловів використовують п'ять логічних зв'язок: $\neg$ (ні), & (і), $\vee$ (або), $\rightarrow$ (якщо ... , то), $\leftrightarrow$ (тоді і тільки тоді).

2.1.1. Логічні зв'язки (операції).

● Заперечення ($\neg F$) – операція, значенням якої є заперечення значення операнда. Тобто $\neg F$ істинна, коли F хибна, і хибна, коли F істинна.

F	$\neg F$
1	0
0	1

Приклад 2. Вислів A = «2 · 2 = 5». Такий вислів хибний, отже A = 0, тобто $\neg A = 1$.

● Кон'юнкція (F_1 & F_2) – операція, значенням якої є істина тоді і тільки тоді, коли істинні обидва вислови F_1 і F_2.

F_1	F_2	F_1 & F_2
0	0	0
0	1	0
1	0	0
1	1	1

На природній мові ця операція виражається сполучними словами: «..і..» «..також..», «як ..,так..», «..незважаючи на..» і тому подібне.

Приклад 3. Нехай є вислови E = «число 6 ділиться на 1, 2, 3» та G = «число 6 є сума дільників 1, 2, 3». Тоді вислів «число 6 ділиться на 1, 2, 3 і представляє суму дільників 1, 2, 3» відображає формула $F = E$ & G.

● Диз'юнкція ($F_1 \vee F_2$) – операція, значенням якої є хибність тоді і тільки тоді, коли хибні обидва вислови F_1 і F_2.

F_1	F_2	$F_1 \vee F_2$
0	0	0
0	1	1
1	0	1
1	1	1

На природній мові ця операція виражається роз'єднувальними словами «..або..», «..чи..» і тому подібне.

Приклад 4. Нехай є вислови: «Я піду в театр» (X_1) і «Я зустріну друга» (X_2). Формула $A = X_1 \vee X_2$ відображає вислів «Я піду в театр або зустріну друга».

● Імплікація ($F_1 \to F_2$) – операція, значенням якої є хибність тоді і тільки тоді, коли істинне значення F_1 і хибне F_2.

F_1	F_2	$F_1 \to F_2$
0	0	1
0	1	1
1	0	0
1	1	1

На природній мові ця операція висловлюється «якщо ..., то ...», «тоді ..., коли ...», «остільки ..., оскільки ...», «за наявності ..., слідує ...», і т. ін. Вислів F_1 називають засновком, а F_2 – висновком.

Приклад 5. Якщо людина не читає книг (A), то вона буде неписьменною (B). Формула даного вислову це $F = A \to C$.

● Еквіваленція ($F_1 \leftrightarrow F_2$) – операція, значенням якої є істина тоді і тільки тоді, коли значення F_1 і F_2 однакові.

F_1	F_2	$F_1 \leftrightarrow F_2$
0	0	1
0	1	0
1	0	0
1	1	1

На природній мові це висловлюється «.. для того, щоб..» , «..необхідно і достатньо..», «..лише за умовою..» і т. ін. Еквіваленція дозволяє виконувати заміщення однієї формули іншою.

Приклад 6. Дані вислови P := «йому потрібний лікар»; S := «він хворий». Тоді формула $F = P \leftrightarrow S$ означає такий вислів: «Йому потрібний лікар, лише за умови, що він хворий».

2.2. Формули алгебри висловів

Пропозиціональними змінними називаються такі змінні, замість яких можна підставляти конкретні вислови. Поняття формули алгебри висловів визначається таким чином:

а) всяка пропозиціональна змінна є формулою;

б) якщо F_1 і F_2 – формули, то вирази $\neg F$, $(F_1 \,\&\, F_2)$, $(F_1 \vee F_2)$, $(F_1 \to F_2)$, $(F_1 \leftrightarrow F_2)$ також є формулами;

в) ніяких формул, крім породжених застосуванням вказаних вище правил, немає.

Зазвичай зовнішні дужки біля формули домовляються не писати. Підформулою формули називається всяка її частина, яка сама є формулою.

2.2.1. Правила запису складних формул

- кожне входження логічної зв'язки «$\neg$» належить до пропозиціональної змінної або формули, що безпосередньо іде слідом за логічною зв'язкою справа;
- кожне входження логічних зв'язок «&»,«$\vee$» після розставлення дужок зв'язує пропозиціональні змінні або формули, що безпосередньо оточують ці логічні зв'язки;
- у формулах немає двох логічних зв'язок, що стоять поряд, – вони повинні бути роз'єднані формулами;
- у формулах немає двох формул, що стоять поряд, – вони повинні бути роз'єднані логічною зв'язкою;
- логічні зв'язки за силою і значущістю впорядковані так: $\neg$, &, $\vee$, $\rightarrow$, $\leftrightarrow$, тобто найсильнішою зв'язкою є заперечення, потім кон'юнкція, диз'юнкція, імплікація і, нарешті, еквіваленція; знання про силу логічних зв'язок дозволяють опускати дужки, без яких і так ясний порядок виконання логічних операцій.

2.2.2. Інтерпретація формул логіки висловів

Інтерпретацією формули називається приписування атомам істиннісних виразів. Припустимо, що P і Q – два атоми, їх істиннісні значення $P = 1$, $Q = 0$. Тоді, істиннісні значення $(\neg P)$, $(P \,\&\, Q)$, $(P \vee Q)$, $(P \rightarrow Q)$ та $(P \leftrightarrow Q)$ дорівнюють 0, 0, 1, 0, 0 відповідно. Істиннісне значення будь-якої формули можливо таким самим чином обчислити, виходячи з істиннісних значень атомів.

Приклад 7. Розглянемо формулу $G = (P \,\&\, Q) \rightarrow (P \leftrightarrow \neg S)$.

Атомами в цій формулі є P, Q і S. Приймемо, що істиннісні значення P, Q і S дорівнюють істина, хибність, істина відповідно. Тоді $(P \,\&\, Q)$ є хибність, оскільки Q хибне; $(\neg S)$ є хибність, оскільки S є істина; $(P \leftrightarrow \neg S)$ є хибність, оскільки P є істина і $(\neg S)$ є істина; та $(P \,\&\, Q) \rightarrow (P \leftrightarrow \neg S)$ є істина, оскільки $(P \,\&\, Q)$ є хибність і $(P \leftrightarrow \neg S)$ є хибність. Таким чином, формула G є істина, якщо атомам P, Q і S приписані значення істина, брехня, істина відповідно.

Приписування істиннісних значень істина, хибність , істина атомам P, Q і S відповідно називається *інтерпретацією формули* G. Оскільки кожному з атомів P, Q і S можна приписати або істина, або хибність, то є $2^3 = 8$ інтерпретацій формули G. У табл. 2.1 дані істиннісні значення формули G для всіх таких 8 інтерпретацій.

Таблиця 2.1 – Істиннісна таблиця формули $G = (P \,\&\, Q) \to (P \leftrightarrow \neg S)$

P	Q	S	$P \,\&\, Q$	$\neg S$	$P \leftrightarrow \neg S$	$(P \,\&\, Q) \to (P \leftrightarrow \neg S)$
0	0	0	0	1	0	1
0	0	1	0	0	1	1
0	1	0	0	1	0	1
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	0	0	1
1	1	0	1	1	1	1
1	1	1	1	0	0	0

Визначення. Кажуть, що формула G *істинна при деякій інтерпретації,* тоді і тільки тоді, коли G набуває значення істина в цій інтерпретації; інакше говорять, що G хибна в цій інтерпретації. Якщо в деякій формулі є n різних атомів, то у цієї формули буде 2^n різних інтерпретацій.

2.2.3. Загальнозначущість та суперечність у логіці висловів

Якщо формула G істинна при всіх інтерпретаціях, то така формула називається *загальнозначущою формулою (або тавтологією)*. Якщо формула G хибна при всіх інтерпретаціях, то така формула називається *суперечливою формулою (або суперечністю)*.

Приклад 8. Розглянемо формулу $G = (P \to Q) \,\&\, P \to Q$.

Атоми цієї формули – P і Q. Отже, формула G має $2^2 = 4$ інтерпретації. Істиннісні значення G при всіх її чотирьох інтерпретаціях дані в табл. 2.

Таблиця 2.2 – Істиннісна таблиця формули $G = (P \to Q) \,\&\, P \to Q$

P	Q	$P \to Q$	$(P \to Q) \,\&\, P$	$(P \to Q) \,\&\, P \to Q$
0	0	1	0	1
0	1	1	0	1
1	0	0	0	1
1	1	1	1	1

Формула G істинна при всіх інтерпретаціях, отже, G – *загальнозначуща формула.*

Приклад 9. Розглянемо формулу $G = (P \to Q) \,\&\, (P \,\&\, \neg Q)$.

Істиннісна таблиця для G дана в табл. 2.3.

Таблиця 2.3 – Істиннісна таблиця формули $G = (P \to Q) \,\&\, (P \,\&\, \neg Q)$

P	Q	$P \to Q$	$\neg Q$	$P \,\&\, \neg Q$	$(P \to Q) \,\&\, (P \,\&\, \neg Q)$
0	0	1	1	0	0
0	1	1	0	0	0
1	0	0	1	1	0
1	1	1	0	0	0

Формула G хибна при всіх інтерпретаціях, отже, G – *суперечлива формула.*

Часто необхідно перетворювати формули з однієї форми в іншу, особливо – в «нормальну форму». Перетворення виконуються шляхом заміни в даній формулі деякої її підформули на деяку формулу, «еквівалентну» замінюваній, і повторенням цієї процедури до тих пір, поки не вийде бажана форма.

Визначення. Говорять, що дві формули F і G *еквівалентні* або що F *еквівалентна* G (позначається $F \cong G$), тоді і тільки тоді, коли істиннісні значення F і G збігаються при кожній інтерпретації F і G.

Приклад 10. Довести еквівалентність формул $P \leftrightarrow Q \cong (P \to Q)\ \&\ (Q \to P)$.

Наведемо таблиці істинності для цих формул:

P	Q	$P \leftrightarrow Q$	$P \to Q$	$Q \to P$	$(P \to Q)\ \&\ (Q \to P)$
1	*2*	*3*	*4*	*5*	*6*
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	1	1	1	1

Якщо порівняти значення логічних функцій в третьому і шостому стовпцях, то можна зробити висновок, що виконання операції еквіваленції завжди можна замістити виконанням операцій імплікації і кон'юнкції.

Приклад 11. Довести еквівалентність формул $P \to Q \cong \neg P \vee Q$.

Наведемо таблиці істинності для цих формул:

P	Q	$P \to Q$	$\neg P$	$\neg P \vee Q$
1	*2*	*3*	*4*	*5*
0	0	1	1	1
0	1	1	1	1
1	0	0	0	0

Операцію імплікації можна замістити виконанням операцій заперечення і диз’юнкції (див. стовпці 3 і 5).

Необхідний достатній запас еквівалентних формул, щоб проводити перетворення формул.

2.3. Розв'язання «логічних» задач

Алгебра висловів може бути з успіхом застосована для вирішення одного типу задач, які називають «логічними». Ці задачі можна вирішувати і безпосереднім міркуванням, але не завжди очевидний шлях таких міркувань, а методи алгебри висловів забезпечують гарантований успіх. У таких задачах, як правило, є ряд висловів, про які відомо, що стільки-то з них істинні, а стільки-то хибні, але не відомо, які саме істинні, а які хибні. Наприклад, є три вислови U, V, W, з яких два істинні, а один хибний. Враховуючи ці умови,

потрібно скласти з цих висловів будь-який складний вислів, який буде свідомо істинний (або хибний). Потім, використовуючи закони логіки, перетворити його до вигляду, з якого визначиться відповідь на питання задачі. У даному прикладі такий складний вислів будується виходячи з таких міркувань. Оскільки з висловів *U, V, W* два істинні, то всі диз'юнкції пар цих висловів також будуть істинні: $U \vee V$, $U \vee W$, $V \vee W$. Отже, буде істинною і кон'юнкція цих висловів: $(U \vee V) \,\&\, (U \vee W) \,\&\, (V \vee W)$. Рівносильне перетворення цього виразу залежатиме від структури висловів *U, V, W*.

Приклад 12. Один з трьох братів Вітя, Толя або Коля розбив вікно. У розмові беруть участь ще двоє братів – Андрій та Діма.

– Це міг зробити тільки або Вітя, або Толя, – сказав Андрій.

– Я вікно не розбивав, – заперечив Вітя, – і Коля теж.

– Ви обидва говорите неправду, – заявив Толя.

– Ні, Толя, один з них сказав правду, а інший сказав неправду, – заперечив Діма.

– Ти, Діма, не маєш рації, – втрутився Коля.

Їх батько, якому, зазвичай, можна довіряти, впевнений, що троє братів сказали правду. Хто розбив вікно?

Рішення. Введемо позначення для висловів:

B: «Вітя розбив вікно»;

T: «Толя розбив вікно»;

K: «Коля розбив вікно».

Тоді вислови братів можна записати в символічній формі таким чином:

$A = B \vee T$;

$V = \neg B \,\&\, \neg K$;

$L = \neg A \,\&\, \neg V = \neg (B \vee T) \,\&\, \neg (\neg B \,\&\, \neg K) = \neg B \,\&\, \neg T \,\&\, (B \vee K) =$

$= \neg T \,\&\, ((\neg B \,\&\, B) \vee (\neg B \,\&\, K)) = \neg T \,\&\, \neg B \,\&\, K$;

$D = (A \,\&\, \neg V) \vee (\neg A \,\&\, V) = ((B \vee T) \,\&\, \neg (\neg B \,\&\, \neg K)) \vee (\neg (B \vee T) \,\&\,$

$\&\, (\neg B \,\&\, \neg K)) = ((B \vee T) \,\&\, (B \vee K)) \vee (\neg B \,\&\, \neg T \,\&\, \neg K) = B \vee (T \,\&\, K) \vee$

$\vee (\neg B \,\&\, \neg T \,\&\, \neg K) = (B \vee \neg B) \,\&\, (B \vee \neg T) \,\&\, (B \vee \neg K) = B \vee (\neg T \,\&\, \neg K)$;

$M = \neg D = \neg (B \vee (\neg T \,\&\, \neg K)) = \neg B \,\&\, (T \vee K)$.

Утворюємо з висловів *A, V, L, D, M* всілякі кон'юнкції по три вислови: $A \,\&\, V \,\&\, L$, $A \,\&\, V \,\&\, D$, $A \,\&\, L \,\&\, D$, $A \,\&\, L \,\&\, M$, $A \,\&\, D \,\&\, M$, $V \,\&\, L \,\&\, D$, $V \,\&\, L \,\&\, M$, $V \,\&\, D \,\&\, M$, $L \,\&\, D \,\&\, M$, $A \,\&\, V \,\&\, M$. Оскільки з висловів *A, V, L, D, M* тільки три істинні, то з десяти кон'юнкцій істинна лише одна. Легко перевірити, що кон'юнкції $A \,\&\, L$, $V \,\&\, L$, $L \,\&\, D$ хибні, тому вісім перерахованих кон'юнкцій помилкові.

Залишаються дві кон'юнкції: $A \,\&\, V \,\&\, D$, $A \,\&\, V \,\&\, M$. Проведемо їх перетворення:

$A \,\&\, V \,\&\, D = (B \vee T) \,\&\, \neg B \,\&\, \neg K \,\&\, (B \vee (\neg T \,\&\, \neg K)) = (B \vee (T \,\&\, \neg T \,\&\,$

$\&\, \neg K)) \,\&\, \neg B \,\&\, \neg K = B \,\&\, \neg B \,\&\, \neg K = 0$;

$A \,\&\, V \,\&\, M = (B \vee T) \,\&\, \neg B \,\&\, \neg K \,\&\, \neg B \,\&\, (T \vee K) = T \,\&\, \neg B \,\&\, \neg K \,\&\, T =$

$= T \,\&\, \neg B \,\&\, \neg K$.

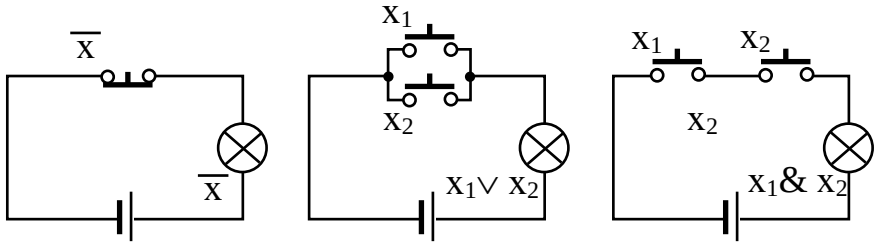
Отже, робимо висновок, що істинним може бути тільки вислів $A \,\&\, V \,\&\, M$ або (еквівалентний йому) $T \,\&\, \neg B \,\&\, \neg K$, тобто істинні вислови T, $\neg B$, $\neg K$, і виходить, що вікно розбив Толя.

2.4. Застосування алгебри логіки в теорії автоматів. Схеми перемикачів

Як одну з інтерпретацій булевих функцій розглянемо електричну схему, що складається з джерела напруги (батареї), лампочки і одного або двох ключів (x_1 і x_2). Ключі управляються кнопками з двома станами: кнопка натиснута (1) і кнопка відпущена (0). Якщо у вихідному стані ключ розімкнений, то при натисненні кнопки він замикається. Ключ може бути сконструйований і так, що в вихідному стані він замкнутий, тоді натиснення кнопки означає його розмикання, тобто приводить до протилежного результату. Тому нормально замкнуті ключі позначимо через $\bar{x}_1$ і $\bar{x}_2$.

При відповідних станах кнопок лампочка приймає один з двох станів: горить (1) і не горить (0). Стани кнопок ототожнюються зі значеннями булевих змінних x_1 та x_2, а стан лампочки – зі значенням функцій цих змінних.

Операції заперечення відповідає схема з одним нормально замкнутим ключем (рис. 2.1, *а*). Якщо кнопка натиснута ($x = 1$), ключ розімкнений і лампочка не горить, тобто $f(x) = 0$; при відпущеній кнопці ($x = 0$) ключ замкнутий і лампочка горить, тобто $f(x) = 1$. Операціям диз'юнкції і кон'юнкції відповідають схеми з двома нормально розімкненими ключами (рис. 2.1, *б*, *в*).



а *б* *в*

Рисунок 2.1 – Схеми перемикачів, відповідні операціям заперечення (*а*), диз'юнкції (*б*) і кон'юнкції (*в*)

Легко переконатися, що в схемі рис. 2.1, *б* лампочка горить при натисненні хоч би однієї з кнопок, а в схемі рис. 2.1, *в* – тільки при натисненні обох кнопок одночасно.

Будь-яку складну булеву функцію можна представити деякою схемою перемикача. На рис. 2.2, *а* показана схема, що реалізує функцію $y = x_1\bar{x}_2 \vee \bar{x}_1 x_2 x_3 \vee x_3 x_4$. Та ж функція представляється рівносильною формулою $y = x_1\bar{x}_2 \vee (\bar{x}_1 x_2 \vee x_4)x_3$, якій відповідає інша простіша схема

(рис. 2.2, *б*). Слід мати на увазі, що ключі, позначені однаковими буквами (*x* або $\bar{x}$), зв'язані між собою і управляються загальною кнопкою.

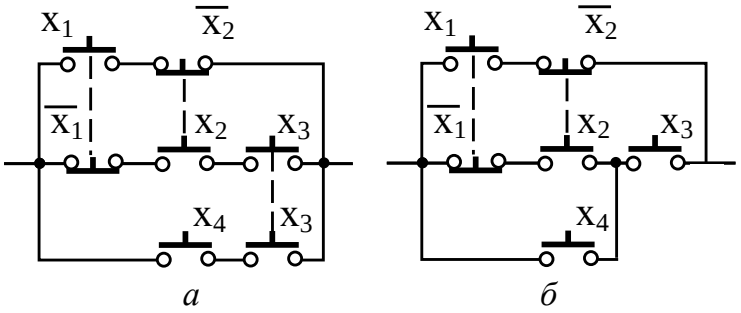


Рисунок 2.2 – Схема перемикача, що реалізує логічну функцію (*а*), та спрощена схема (*б*)

У реальних пристроях використовуються ключі різної конструкції і фізичної природи (механічні, електромагнітні, електронні, гідравлічні, пневматичні і так далі). Проте при реалізації логічних функцій багато технічних особливостей не мають значення. Істотними властивостями контактних схем є початкові положення ключів (нормально розімкнені або нормально замкнуті) та спосіб їх з'єднання між собою і зовнішніми пристроями. Ця інформація повністю відображається графом, ребра якого відповідають ключам, а вершини – точкам їх з'єднання. Ребра нормально розімкнених ключів позначаються відповідною змінною (*x*), а нормально замкнутих – запереченням змінної $(\bar{x})$. Наприклад, контактна схема (рис. 2.2, *б*) зображається графом, як показано на рис. 2.3, *а*.

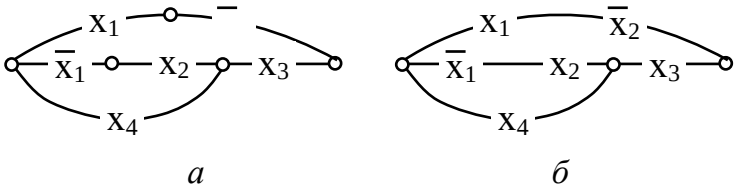


Рисунок 2.3 – Граф схеми перемикача (*а*) та його спрощене зображення (*б*)

При зображенні контактних схем графами приймаються деякі специфічні умови і спрощення. Зазвичай змінні позначаються в розривах ліній, що зображають ребра. При цьому ребрами вважаються тільки такі лінії, які позначені якою-небудь змінною або її запереченням. Інші лінії, що не є ребрами графа, можуть зображати входи і виходи схеми, зв'язки з іншими схемами і тому подібне. Крім того, вершини другого степеня можуть не зображатися, оскільки їм інцидентні пари послідовно з'єднаних ребер, з яких кожне позначене відповідною змінною. На рис. 2.3, *б* показана контактна схема у загальноприйнятому вигляді.

2.5. Логіка першого порядку (логіка предикатів). Загальні поняття

Початкові елементи в логіці висловів називаються атомами. З атомів будуються формули. Атом являє собою оповідне речення, яке може бути або істинним або хибним, але не те й інше разом. Атом розглядається як єдине ціле. Його структура і склад не аналізуються.

Часто вислови виражають властивості одного або декількох об'єктів. Змістовна частина вислову відіграє роль визначальної властивості сукупності об'єктів, для яких цей вислів істинний, і називається *предикатом (або висловлювальною функцією*). Наприклад, вислів «Іванов – відмінник» істинний або помилковий залежно від оцінок, які має даний студент. У той же час предикат «x – відмінник» визначає підмножину відмінників на деякій множині студентів (група, курс, факультет). Підставивши замість x прізвища студентів, отримаємо множину висловів. Сукупність дійсних висловів і відповідатиме підмножині відмінників.

Предикат є логічною функцією $P(x)$, що приймає, як і булеві функції, значення 0 або 1, але на відміну від них, значення аргументу x вибираються з деякої множини M об'єктів $\{x \in M\}$.

Приклад 13. Представити твердження «x більше 3». Спочатку визначається предикат *БІЛЬШЕ* (x, y), який означає «x більше y». Тоді вираз «x більше 3» представляється виразом *БІЛЬШЕ* (x, 3).

У логіці першого порядку можна використовувати *функціональні символи*. Наприклад, можемо використовувати *плюс* (x, y), щоб позначити «$x + y$». Пропозицію «x + 1 більше x» можна символічно представити у вигляді *БІЛЬШЕ* (*плюс* (x, 1), x).

У наведеному прикладі вирази *БІЛЬШЕ* (x, 3), *БІЛЬШЕ* (*плюс* (x, 1), x) є атомами логіки першого порядку, де *БІЛЬШЕ – предикативний символ*; x – *змінна*; 3 – *константа*; *плюс – функціональний символ*.

Якщо висловлювальна функція містить один аргумент, то заданий *одномісний предикат*, якщо вона містить n аргументів, то – *n-місний предикат*. Одномісний предикат, як правило, описує наявність будь-якої ознаки у предмета, а n-місний предикат – наявність відношень між n предметами.

Для змінних використовуються два спеціальні символи – $\forall$ і $\exists$. Символи $\forall$ і $\exists$ називаються відповідно кванторами спільності та існування. Якщо x – змінна, то ($\forall x$) читається як «*для всіх x*», «*для кожного x*» або «*для всякого x*», тоді як ($\exists x$) читається «*існує x*», «*для деяких x*» або «*принаймні для одного x*».

Множина предметних змінних $T_1 = \{x, y, ...\}$, предметних постійних $T_2 = \{a, b,..\}$, функціональних символів $T_3=\{f_1, f_2, ...\}$ і предикативних символів $T_4 = (P_1, P_2, ...\}$ із заданими над множиною $T = \{T_1, T_2, T_3, T_4\}$ операціями $F = \{\neg; \&; \vee; \rightarrow; \leftrightarrow; \forall; \exists\}$ формують *алгебру предикатів*, тобто $A_n = <T; F>$.

Будь-яку предметну змінну і предметну постійну в алгебрі предикатів називають *термом*.

Приклад 14. Записати такі твердження:

а) «Кожне раціональне число є дійсне число».

Позначимо «x є просте число» через $P(x)$, «x є раціональне число» через $Q(x)$. Тоді твердження може бути записане виразом

$$(\forall x)(Q(x) \to R(x)).$$

б) «Для кожного числа x існує таке число y, що $x < y$».

Позначимо «x менше y» через *МЕНШЕ* (x, y). Тоді твердження може бути записане виразом

$$(\forall x)(\exists y)\ \textit{МЕНШЕ}\ (x, y).$$

2.6. Інтерпретація формул логіки предикатів

У логіці висловів інтерпретація є приписування атомам істиннісних значень. Щоб визначити інтерпретацію для формули логіки першого порядку, необхідно вказати предметну область (область значень предметних *змінних*) і значення *констант*, *функціональних* і *предикативних символів*, що зустрічаються у формулі.

Визначення. Інтерпретація формули F логіки першого порядку складається з непорожньої (предметної) області D і зазначення «оцінки» (значення) всіх констант, функціональних символів і предикативних символів, що зустрічаються в F.

Коли шукається «оцінка», тобто визначається істиннісне значення формули в інтерпретації на області D, $(\forall x)$ інтерпретуватиметься як «для всіх елементів x з D» і $(\exists x)$ – як «існує елемент x з D».

Для кожної інтерпретації формули на області D формула може набути істиннісного значення 1 або 0 згідно з таким правилом:

1. Якщо задані значення формул G і H,то істиннісні значення формул $\neg G$, $(G \,\&\, H)$, $(G \vee H)$, $(G \to H)$ і $(G \leftrightarrow H)$ отримуються за допомогою таблиць істинності з логіки висловів.

2. $(\forall x)G$ набуває значення *істина*, якщо G набуває значення *істина* для кожного x з D; інакше вона набуває значення *хибність*.

3. $(\exists x)G$ набуває значення *істина*, якщо G набуває значення *істина* хоч би для одного x з D; інакше вона набуває значення *хибність*.

Приклад 15. Розглянути формули

$$(\forall x)P(x) \quad \text{і} \quad (\exists x)\neg P(x).$$

Нехай їхня інтерпретація така:

Область: $D = \{1,2\}$.

Оцінка для P:

$P(1)$	$P(2)$
1	0

Легко переконатися, що $(\forall x)P(x)$ є *хибність* в цій інтерпретації тому, що за правилом 2 $P(x)$ повинне набувати значення *істина* для кожного x з D, а тут $P(2)$ = *хибність*.

Формула $(\exists x)\neg P(x)$ – *істина* в цій інтерпретації, оскільки за правилом 3 хоч би для одного x з D $\neg P(x)$ повинне набувати значення *істина,* тут $\neg P(2) =$ $=$ *істина.*

Приклад 16. Розглянути формулу

$$(\forall x)(\exists y)\ \mathrm{P}(x, y).$$

Інтерпретація визначена таким чином:

Область: $D = \{1,2\}$.

Оцінка для P:

$P(1,1)$	$P(1,2)$	$P(2,1)$	$P(2,2)$
1	0	1	0

Якщо $x = 1$, можна бачити, що існує такий x (а саме 1), що $P(1, y)$ є *істина*.

Якщо $x = 2$, також існує такий y, що $P(2, y)$ є *істина* .

Отже, у вказаній інтерпретації для кожного x з D існує такий y, що $P(x, y)$ є *істина*, тобто $(\forall x)(\exists y)\ P(x, y)$ є істина в цій інтерпретації.

Приклад 17. Розглянути формулу

$$(\forall x)\ (P(x) \to Q(\ f(x), a)).$$

У формулі є одна константа a, один одномісний функціональний символ f, один одномісний предикативний символ P і один двомісний предикативний символ Q. Нижче наводиться інтерпретація формули.

Область: $D = \{1, 2\}$.

Оцінки для a: $a = 1$.

Оцінки для f:

$f(1)$	$f(2)$
2	1

Оцінки для P і Q:

$P(1)$	$P(2)$	$Q(1,1)$	$Q(1,2)$	$Q(2,1)$	$Q(2,2)$
0	1	1	1	0	1

Якщо $x = 1$, то

$P(x) \to Q(f(x), a) = P(l) \to Q(f(l), a) = P(l) \to Q(2, 1) = 0 \to 0 = 1.$

Якщо $x = 2$, то

$P(x) \to Q(\ f(x), a) = P(2) \to Q(f(2), a) = P(2) \to Q(1, 1) = 1 \to 1 = 1.$

Оскільки $P(x) \to Q(\ f(x), a)$ істинно для всіх елементів x з області D, то формула $(\forall x)\ (P(x) \to Q(\ f(x), a))$ є *істина* в даній інтерпретації**.**

2.7. Передуюча нормальна форма

Для зручності аналізу складних висновків рекомендується формули приводити до нормальної форми. Якщо в алгебрі висловів прийнято дві нормальні форми (ДНФ – диз'юнктивна і КНФ – кон'юнктивна), то в алгебрі

предикатів – одна *передуюча нормальна форма* (ПНФ), суть якої зводиться до розділення формули на дві частини: кванторну і безкванторну. Для цього квантори формули виносять вліво за певними правилами алгебри предикатів.

У результаті таких перетворень може бути отримана формула вигляду: $\mathcal{R}_{x1}\ \mathcal{R}_{x2}\ \ldots \mathcal{R}_{xn}(M)$, де $\mathcal{R} \in \{\forall;\ \exists\}$, а M – матриця формули. Кванторну частину формули $\mathcal{R}_{x1}\ \mathcal{R}_{x2}\ \ldots \mathcal{R}_{xn}$ іноді називають префіксом формули. Потім матрицю формули перетворюють до вигляду КНФ.

Ці формули знаходяться в передуючій нормальній формі:

$$(\forall x)(\forall y)\ (P(x, y)\ \&\ Q(y)),$$
$$(\forall x)(\forall y)\ (\neg P(x, y) \to Q(y)),$$
$$(\forall x)(\forall y)\ (\exists z)\ (Q(x,y) \to R(z)).$$

2.7.1. Перетворення формул в передуючу нормальну форму

Крок 1. Використовують закони

$$F \leftrightarrow G = (F \to G)\ \&\ (G \to F),$$
$$F \to G = \neg F \vee G,$$

щоб виключити логічні зв'язки $\leftrightarrow$ і $\to$.

Крок 2. Повторно використовують закон

$$\neg(\neg F) = F,$$

закони Моргана

$$\neg(F \vee G) = \neg F\ \&\ \neg G,$$
$$\neg(F\ \&\ G) = \neg F \vee \neg G.$$

і закони

$$\neg(\forall x)(F)=(\exists x)(\neg F),$$
$$\neg(\exists x)(F)=(\forall x)(\neg F),$$

щоб перенести знак заперечення всередину формули.

Крок 3. Використовують закони

$$(\forall x)\ F\ \&\ (\forall x)\ G = (\forall x)\ (F\ \&\ G),$$
$$(\exists x)\ F\ \vee (\exists x)\ G = (\exists x)\ (F\ \vee G),$$

щоб винести квантори в самий початок формули для отримання формули, що знаходиться в передуючій нормальній формі.

Приклад 18. Привести формулу $(\forall x)P(x) \to (\exists x)Q(x)$ до передуючої нормальної форми:

$(\forall x)P(x) \to (\exists x)Q(x) = \neg((\forall x)P(x)) \vee (\exists x)Q(x) =$

$= (\exists x)(\neg P(x)) \vee (\exists x)Q(x) = (\exists x)(\neg P\ (x) \vee Q(x)).$

Отже, передуюча нормальна форма формули $(\forall x)P(x) \to (\exists x)Q(x)$ це – $(\exists x)(\neg P\ (x) \vee Q(x))$.

Приклад 19. Отримати передуючу нормальну форму для формули

$$(\forall x)(\forall y)((\exists z)\ P(x, z)\ \&\ P(y, z)) \to (\exists u)\ Q(x, y, u)).$$

$(\forall x)(\forall y)((\exists z)\ P(x, z)\ \&\ P(y, z)) \to (\exists u)\ Q(x, y, u)) =$

$= (\forall x)(\forall y)(\neg((\exists z)\ P\ (x, z)\ \&\ P\ (y, z))) \vee (\exists u)\ Q(x, y, u)) =$

$= (\forall x)(\forall y)((\exists z)\ (\neg P\ (x, z) \vee \neg P\ (y, z)) \vee (\exists u)\ Q(x, y, u)) =$
$= (\forall x)(\forall y)(\forall z)(\exists u)\ (\neg P\ (x, z) \vee \neg P\ (y, z) \vee Q(x, y, u)).$
Остання формула є передуючою нормальною формою першої формули.

2.8. Логіка реляційна

Двовимірна таблиця є найбільш зручною формою для подання, пошуку і обробки інформації. Якщо іменами стовпців таблиці є імена яких-небудь ознак, які найчастіше називаються *атрибутами*, а рядками – ланцюжки значень заданих атрибутів, які найчастіше називаються *кортежами*, то множину таких ланцюжків таблиці називають *відношенням*. Всі кортежі відношення мають бути сумісними, тобто мати одне і те ж число атрибутів, а значення одного атрибуту в кожному кортежі відношення повинне вибиратися з однієї області визначення, названої доменом. Число стовпців таблиці або атрибутів відношення визначає його арність, а число кортежів – потужність відношення.

Реляційна модель бази даних – це остаточний набір скінченних відношень, що спираються на різні, але кінцеві набори атрибутів. Над відношеннями реляційної моделі можна здійснювати різні алгебраїчні та логічні операції для дослідження характеристик об'єктів або зв'язків між об'єктами. Тим самим реляційна модель стала областю застосування математичної логіки.

Відношення можна розглядати як *файл* певного типу. Такий файл складається з послідовності *записів*, по одному на кожен кортеж, причому не повинно бути однакових записів. Всі записи повинні мати однакове число *полів* для представлення атрибутів. У однойменних полях різних записів повинна зберігатися інформація одного і того ж типу, відповідного заданому, типу атрибуту.

Між таблицею, відношенням і файлом є відповідність:

ТАБЛИЦЯ	ВІДНОШЕННЯ	ФАЙЛ
рядок	кортеж	запис
ім'я стовпця	ім'я атрибуту	ім'я поля
тип атрибуту	тип домена	тип поля

Верхній рядок таблиці формує головку, а решта рядків – вміст таблиці:

Дисципліна	Лекції (год)	Лаб. заняття (год)	Практ. заняття (год)	Звітність (залік, екз.)
фізика	34	34	17	экз.
інформатика	51	34	0	залік

Це відношення відображає зв'язки між найменуванням навчальної дисципліни, видом і числом годин аудиторних занять та формою звітності. Імена атрибутів відношення: «Дисципліна», «Лекції_(год.)», «Лаб. заняття_(год.)»,

«Практ. заняття_(год.)», «звітність_(залік, екз.)». Значенням атрибуту «Дисципліна» є слова {фізика, інформатика}, «звітність» – {залік, екз.}. Тип цих атрибутів – <слово>, а тип домену – *CHAR* (строкове). Значеннями атрибутів «Лекції_(год.)», «Лаб. заняття_(год.)» і «Практ. заняття_(год.)» є цілі числа {17, 34, 51, 68}. Тип цих атрибутів – <ціле_число>, а тип домену – *INTEGER* (ціле). Кортежами цього відношення є: (фізика, 34, 34, 17, екз.), (інформатика 51, 34, 0, залік).

Таким чином, якщо дана множина атрибутів $A = \{A_1, A_2, ..., A_n\}$ і множина доменів $D = \{D_1, D_2, ..., D_m\}$, то кортеж відношення є $t = (d_1, d_2, ..., d_n)$, де $d_i \in D$.

Відношення є множиною сумісних кортежів, тобто

$$r = \{t | \ t = (d_1, d_2, ..., d_n),\ d_i \in D\}.$$

Відношення, задане на множині впорядкованих кортежів, є підмножиною n-арного прямого добутку домену D, тобто

$$r = \{t | \ t=(d_1, d_2,...d_n),\ d_i \in D\} \subseteq {}_1\otimes^n D.$$

Відношення на множині впорядкованих кортежів задають *схемою відношення* із зазначенням імені відношення, числа і послідовності атрибутів кортежу $rel(r)$, тобто $rel(r) = (A_1, A_2, \ldots, A_n)$.

Множина схем відношень, що використовуються в реляційній базі даних, називають *схемою реляційної бази даних*, тобто $REL(R) = \{rel(r)\}$.

2.9. Нечітка логіка. Загальні поняття

Часто потрібно приймати рішення при неповному, нечіткому або неясному описі об'єктів або зв'язків між ними. Наприклад, «великий вхідний опір осцилографа», «постійне число обертів двигуна» не дають числової характеристики атрибута об'єкта, або «судно стоїть біля причалу», «літак знаходиться в аеропорту» не дають числової оцінки зв'язків між об'єктами, або «якщо йде дощ, то закриті всі люки», «якщо температура тіла 38 °С, то людина хвора» дають лише якісний опис логіки рішень.

Неповний опис елемента або ситуації не дозволяє оцінити міру їх приналежності до певного класу і з упевненістю дати оцінку правильності рішення.

Для подібних задач розроблена нечітка логіка (*fuzzi logic*), об'єктом дослідження якої є нечіткі множини (*fuzzi set*), нечіткі відношення (*fuzzi relation*), нечіткі операції алгебри і нечітке числення (*fuzzi calculus*).

Звичайна «чітка» множина має чіткі межі в деякому універсумі. Приналежність елемента універсуму до деякої множини оцінюється бінарно: «Так» чи «Ні». Така приналежність елемента x до множини A на універсумі U може бути описана функцією приналежності μ_A

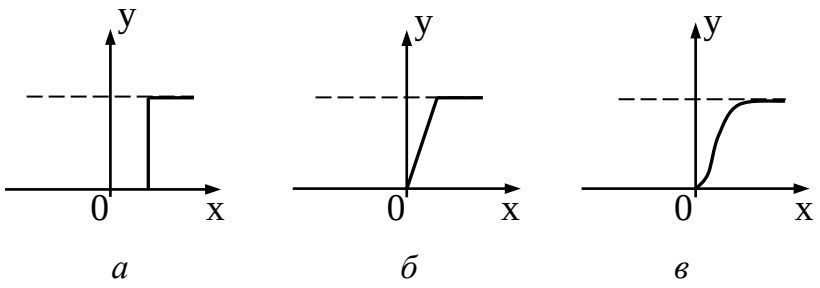
$$A = \{(x, \mu_A(x))\},\ x \in U,\ \mu_A(x) \in \{0,1\}.$$

Тут A задано як сукупність пар – «елемент – степінь приналежності до множини», тобто для звичайної множини μ_A – це відображення елементів універсуму U в бінарну множину $\{0, 1\}$: $\mu_A: U \rightarrow \{0, 1\}$.

Нечітка множина A в універсумі (просторі) U задається функцією приналежності, що відображає універсум не в бінарну множину, а в інтервал [0,1]. Тоді задання нечіткої множини виглядає так:

$$A = \{(x, \mu_A(x))\},\ x \in U,\ \mu_A(x) \in [0,1],\ :\mu_A: U \to [0, 1],$$

тобто степінь приналежності елемента до множини оцінюється не стрибком «або 0, або 1», а плавно, наприклад: 0; 0,1; 0,4; 0,7; 0,9; 1. Функція приналежності може задаватися формулою або графічно (наприклад так, як вказано на рис. 2.4).



а *б* *в*

Рисунок 2.4 – Функції приналежності:
а – дискретна; *б* – плавно-кусково-лінійна; *в* – сигмоїда

У випадку кінцевої осяжної множини застосовують такий запис:

$$A = \left\{ \frac{0}{1} + \frac{0,1}{2} + \frac{0,3}{3} + \frac{0,5}{4} + \frac{0,8}{5} \right\}.$$

Це означає, що:

- елемент 1 належить до універсуму U зі степенем 0, тобто не належить;
- елемент 2 належить до універсуму U зі степенем 0,1;
- елемент 3 належить до універсуму U зі степенем 0,3;
- елемент 4 належить до універсуму U зі степенем 0,5;
- елемент 5 належить до універсуму U зі степенем 0,8.

Знак + означає об'єднання елементів. Сам універсум $U = 1 + 2 + 3 + 4 + 5$.

Множина порожня, якщо $\forall x \in U\ \mu_A(x) = 0$, тобто $A = \varnothing$.

Якщо $\forall x \in U\ \mu_A(x) = 1$, то $A = U$.

Дві множини A і B рівні, якщо $\mu_A = \mu_B$, тобто $A = B$.

Множина A включається у B, якщо $\mu_A \le \mu_B$, тобто $A \subseteq B$.

Множина $\overline{A}$ є доповненням множини A, якщо $\mu_A = 1 - \mu_A$.

Вводяться також двомісні теоретико-множинні операції.

Приклад 20. Нехай дано 10 дискет. Множина всіх підмножин цих дискет містить порожню множину, одно-, дво- три- і так далі до десятиелементної підмножини. Так, задано універсальну множину $U = \{\varnothing, 1, 2, 3, ..., 10\}$. На цій множині необхідно вибрати підмножини, що задовольняють поняттю «Вибрати декілька дискет».

Для підмножин, що містять нуль, один, два, половину або всі дискети, експерт визначив значення функції приналежності рівним нулю, оскільки можна було б сказати конкретно: «взяти половину дискет», «взяти дві

дискети» і тому подібне Для підмножин, що містять три, вісім або дев'ять дискет, експерт визначив значення функції приналежності рівним 0,6, а для підмножин, що містять чотири, шість або сім дискет, – рівним 0,8.

Отже, експерт виконав поставлене завдання так:

$$X = \left\{ \frac{0,6}{3} + \frac{0,8}{4} + \frac{0,8}{6} + \frac{0,8}{7} + \frac{0,6}{8} + \frac{0,6}{9} \right\}$$

Носієм цієї нечіткої підмножини є $X = \{3, 4, 6, 7, 8, 9\}$.

Така була суб'єктивна міра приналежності кожної підмножини універсальної множини нечіткому поняттю «Узяти декілька дискет».

2.10. Нечітка алгебра

1. Перетином нечітких множин A і B є множина C, що складається зі всіх тих елементів базової множини U, які належать і нечіткій множині A і нечіткій множині B

$$C = (A \cap B).$$

Степінь приналежності елемента базової множини нечіткій множині C дорівнює мінімальному значенню функції приналежності для нечітких множин A і B, тобто

$$\mu_{A \cap B} = \min\{\mu_A; \mu_B\}.$$

Наприклад:

$$A = \left\{ \frac{0}{1} + \frac{0,1}{2} + \frac{0,5}{4} + \frac{0,8}{5} \right\}, \quad B = \left\{ \frac{1}{1} + \frac{0,9}{2} + \frac{0,5}{3} + \frac{0,4}{4} \right\},$$

$$A \cap B = \left\{ \frac{0}{1} + \frac{0,1}{2} + \frac{0,4}{4} \right\}.$$

2. Об'єднанням нечітких множин A і B є множина C, що складається зі всіх тих елементів множини U, які належать хоч би одній нечіткій множині A або B

$$C = (A \; U \; B).$$

Степінь приналежності елемента базової множини нечіткій множині C дорівнює максимальному значенню функції приналежності для нечітких множин A і B, тобто

$$\mu_{AUB} = \max\{\mu_A; \mu_B\}.$$

Наприклад:

$$A = \left\{ \frac{0,6}{1} + \frac{0,4}{2} + \frac{0,5}{4} + \frac{1,0}{5} \right\}, \quad B = \left\{ \frac{0,9}{1} + \frac{0,4}{2} + \frac{1,0}{3} + \frac{0,5}{9} \right\},$$

$$A \cup B = \left\{ \frac{0,9}{1} + \frac{0,4}{2} + \frac{1,0}{3} + \frac{0,5}{4} + \frac{1,0}{5} + \frac{0,5}{9} \right\}.$$

3. Доповненням нечіткої множини A є нечітка множина $\neg A$, що складається зі всіх елементів універсальної множини U, які не належать до нечіткої множини A.

Степінь приналежності елемента нечіткій множині $\neg A$ дорівнює доповненню до значення степеня приналежності базовій множині U, тобто

$$\mu_{\neg A}(u) = 1 - \mu_A(u).$$

Наприклад:

$$A = \left\{ \frac{0,6}{1} + \frac{0,4}{2} + \frac{0,8}{3} + \frac{0,2}{4} + \frac{1,0}{5} + \frac{0,3}{6} \right\},$$

$$B = \left\{ \frac{0,9}{1} + \frac{0,4}{2} + \frac{1,0}{3} + \frac{0,7}{7} + \frac{0,3}{8} + \frac{0,5}{9} \right\}.$$

Для заданих множин маємо:

$$\neg A = \left\{ \frac{0,4}{1} + \frac{0,6}{2} + \frac{0,2}{3} + \frac{0,8}{4} + \frac{0,7}{6} + \frac{1,0}{7} + \frac{1,0}{8} + \frac{1,0}{9} \right\},$$

$$\neg B = \left\{ \frac{0,1}{1} + \frac{0,6}{2} + \frac{1,0}{4} + \frac{1,0}{5} + \frac{1,0}{6} + \frac{0,3}{7} + \frac{0,7}{8} + \frac{0,5}{9} \right\}.$$

4. Різницею нечітких множин A і B є множина C, що складається з тих елементів множини U, які належать нечіткій множині A і не належать нечіткій множині B, тобто

$$C = A \setminus B = A \cap \neg B.$$

Степінь приналежності нечіткій множині C дорівнює мінімальному значенню функції приналежності одного і того ж елемента нечітких множин A і $\neg B$, тобто

$$\mu_C(u) = \mu_A(u) \,\&\, (1 - \mu_B(u)) = \min \{\mu_A(u); (1 - \mu_B(u))\}.$$

Наприклад: Для множин з попереднього прикладу маємо:

$$C = A \setminus B = \left\{ \frac{0,1}{1} + \frac{0,4}{2} + \frac{0,2}{4} + \frac{1,0}{5} + \frac{0,3}{6} \right\}.$$

5. Симетричною різницею нечітких множин A і B є множина C, що складається зі всіх тих елементів універсальної множини U, які належать нечіткій множині A і не належать нечіткій множині B або належать нечіткій множині B і не належать нечіткій множині A, тобто

$$C = A \oplus B = (A \cap \neg B) \cup (B \cap \neg A).$$

Степінь приналежності нечіткій множині C дорівнює максимальному значенню двох мінімальних значень одного і того ж елемента множин $(A \cap \neg B)$ і $(B \cap \neg A)$, тобто

$$\mu_C(u) = (\mu_A(u) \,\&\, \mu_{\neg B}(u)) \vee (\mu_B(u) \,\&\, \mu_{\neg A}(u_i)) =$$
$$= \max \{\min \{\mu_A(u); \mu_{\neg B}(u)\}; \min \{\mu_B(u); \mu_{\neg A}(u_i)\}\}.$$

Наприклад: Для множин з попереднього прикладу маємо:

$$C = A \oplus B = \left\{ \frac{0,4}{1} + \frac{0,4}{2} + \frac{0,2}{3} + \frac{0,2}{4} + \frac{1,0}{5} + \frac{0,3}{6} + \frac{0,3}{7} + \frac{0,3}{8} + \frac{0,5}{9} \right\}.$$

6. Прямим добутком нечітких множин A і B є множина C, що складається зі всіх тих або тільки тих впорядкованих пар (u_i, u_j), перша компонента яких належить множині A, а друга – множині B, тобто

$$C = A \otimes B.$$

Степінь приналежності (u_i, u_j) нечіткій множині C дорівнює мінімальному значенню функцій приналежності елементів $u_i \in A$ і $u_j \in B$:

$$\mu_C(u_i, u_j) = (\mu_A(u_i) \,\&\, \mu_B(u_j) = \min\{\mu_A(u_i);\ \mu_B(u_j)\}.$$

Наприклад: Для множин з попереднього прикладу маємо матрицю суміжності:

C	$u_j = u_1$	$u_j = u_2$	$u_j = u_3$	$u_j = u_7$	$u_j = u_8$	$u_j = u_9$
$u_1 = u_i$	0,6	0,4	0,6	0,6	0,3	0,5
$u_1 = u_i$	0,4	0,4	0,4	0,4	0,3	0,4
$u_3 = u_i$	0,8	0,4	0,8	0,7	0,3	0,5
$u_4 = u_i$	0,2	0,2	0,2	0,2	0,2	0,2
$u_5 = u_i$	0,9	0,4	1,0	0,7	0,3	0,5
$u_6 = u_i$	0,3	0,3	0,3	0,3	0,3	0,3

2.11. Нечітке числення

Нечіткими висловами є речення A, степінь істинності $\rho(A)$ або хибності $\neg\rho(A)$ яких також набуває значення на інтервалі [0; 1]. Наприклад, вислів: «Сьогодні гарна погода». За якими ознаками і хто дав таку оцінку?

Нечіткими предикатами є висловлювальні функції, аргументами яких є предметні змінні. Степінь істинності наочних змінних і висловлювальних функцій також належить інтервалу [0; 1].

Наприклад, вислів «Петров виконує відповідальне завдання». Як розуміти «відповідальне завдання»?

Нечіткі формули. Для формування складних висловів використовують логічні зв'язки заперечення, кон'юнкції, диз'юнкції, імплікації та еквіваленції. Так формуються *нечіткі логічні формули.*

Степінь істинності складного вислову визначається як степінь приналежності результатів виконання операцій над нечіткими множинами:

$\rho(\neg A) = (1 - \rho(A))$;

$\rho(A \,\&\, B) = \min\{\rho(A); \rho(B)\}$;

$\rho(A \vee B) = \max\{\rho(A); \rho(B)\}$;
$\rho(A \to B) = \max\{(1 - \rho(A)); \rho(B)\}$;
$\rho(A \leftrightarrow B) = \min\{\max\{(1 - \rho(A)); \rho(B)\}; \max\{(1 - \rho(B)); \rho(A)\}\}$.

Слід звернути увагу на те, що закони суперечності і «третього не дано» для нечітких висловів не виконуються.

Так для чітких висловів: $A \,\&\, \neg A = 0$, $A \vee A = 1$, а для нечітких висловів:

$$\rho(A \,\&\, \neg A = \min\{\rho(A); (1 - \rho(A))\},\ \rho(A \vee \neg A) = \max\{\rho(A); (1 - \rho(\neg A))\}.$$

Нечіткі правила висловів. Імплікація в тому вигляді, який використовується в класичній логіці, не застосовується. Найчастіше використовують імплікацію у вигляді: «Якщо *A*, то *B*, інакше *C*».

Очевидно, що нечіткий вислів «якщо *A*, то *B*» можна визначити як нечітке відношення між нечіткими висловами *A* і *B*, тобто $(A \otimes B)$, а нечіткий вислів «якщо не *A*, то *C*» – як нечітке відношення між висловами $\neg A$ і *C*. Об'єднанням цих двох відношень є формула умовного нечіткого вислову:

$$((A \to B), C) = ((A \otimes B) (\neg A \otimes C)).$$

Якщо надані значення степенів істинності нечітких висловів $\rho(A)$, $\rho(B)$ і $\rho(C)$, то істинність вислову «якщо *A*, то *B* інакше *C*» може бути визначена, як для нечітких відношень, за формулою:

$$\rho((A \to B), C) = \max\{\min\{\rho(A), \rho(B)\}; \min\{\rho(\neg A), \rho(C)\}\}.$$

Приклад 21. «Якщо сьогодні ввечері буде дощ, то завтра буде сонячна погода, інакше завтра буде похмурий день». Для вислову A = «сьогодні ввечері буде дощ» прийнято $\rho(A) = 0{,}3$, для вислову B = «завтра буде сонячна погода» – $\rho(B) = 0{,}5$, а для вислову C = «завтра буде похмурий день» – $\rho(C) = 0{,}2$.

Отже

$$\rho((A \to B), C) = \max\{\min\{\rho(A), \rho(B)\}; \min\{\rho(\neg A), \rho(C)\}\} =$$
$$= \max\{\min\{0{,}3; 0{,}5\}; \min\{0{,}7; 0{,}2\} = \max\{0{,}3; 0{,}2\} = 0{,}3.$$

Якщо $\rho(C) = 1$, тобто вислів *C* є істиною для будь-яких значень істинності вислову $\neg A$, то формула умовного вислову набирає вигляду, $((A \otimes B) \vee \neg A)$, що відповідає вислову «якщо *A*, то *B*».

Степінь істинності такого вислову є

$$\rho(A \to B) = \max\{\min\{\rho(A), \rho(B)\}; \rho(\neg A)\}.$$

Так можна визначити істинність імплікації за відомими значеннями істинності засновку *A* і висновку *B*.

Приклад 22. «Якщо сьогодні ввечері буде дощ, то завтра буде сонячна погода». Для вислову A = «сьогодні ввечері буде дощ» прийнято $\rho(A) = 0{,}3$, для вислову B = «завтра буде сонячна погода» прийнято $\rho(B) = 0{,}5$.

Отже

$$\rho(A \to B) = \max\{min\{\rho(A), \rho(B)\}; \rho(\neg A)\} =$$
$$= \max\{\min\{0{,}3, 0{,}5\}; 0{,}7\} = \max\{0{,}3; 0{,}7\} = 0{,}7.$$

Якщо надані множини нечітких висловів $\{A = \rho(u_i)/u_i\}$ і $\{B = \rho(v_j)/v_j\}$ про факти $\{u_1, u_2, u_3, u_4, u_5, u_6\}$ і $\{v_1, v_2, v_3, v_4, v_5, v_6\}$, то істинність $\rho(A \to B)$ необхідно визначати для кожної пари (u_i, v_j) за формулою:

$$\rho(A \to B) = \max\{\min\{\rho(u_i), \rho(v_j)\}; \neg\rho(u_i)\}.$$

Приклад 23. Нехай надані нечіткі вислови

$$\rho(A) = \left\{\frac{0{,}6}{u_1}; \frac{0{,}4}{u_2}; \frac{0{,}8}{u_3}; \frac{0{,}2}{u_4}; \frac{1{,}0}{u_5}; \frac{0{,}3}{u_6}\right\},$$

$$\rho(B) = \left\{\frac{0{,}9}{v_1} + \frac{0{,}4}{v_2} + \frac{1{,}0}{v_3} + \frac{0{,}7}{v_4} + \frac{0{,}3}{v_5} + \frac{0{,}5}{v_6}\right\}.$$

Для кожної позиції таблиці $\rho(A \to B)$ потрібно обчислити значення $\rho(u_i \to v_j) = \max\{\min\{\rho(u_i), \rho(v_j)\}; \neg\rho(u_i)\}$.

Наприклад $\rho(u_4 \to v_2) = \max\{\min\{0{,}2; 0{,}4\}; 0{,}8\} = \max\{0{,}2; 0{,}8\} = 0{,}8$.

Всі результати обчислень $\rho(u_i \to v_j)$ зведені в таблицю:

	v_1	v_2	v_3	v_4	v_5	v_6
u_1	0,6	0,4	0,6	0,6	0,4	0,5
u_2	0,6	0,6	0,6	0,6	0,6	0,6
u_3	0,8	0,4	0,8	0,7	0,3	0,5
u_4	0,8	0,8	0,8	0,8	0,8	0,8
u_5	0,9	0,4	1,0	0,7	0,3	0,5
u_6	0,7	0,7	0,7	0,7	0,7	0,7

Контрольні запитання

1. Які з цих речень є висловами:
   а) Місяць є супутником Марса;
   б) 2 + 2 – 5;
   в) кисень – газ;
   г) залізо легше за воду.
2. Які з висловів попереднього завдання істинні, а які хибні?
3. Визначити значення істинності таких висловів:
   а) Санкт-Петербург розташований на Неві та 2 + 3 = 5;
   б) 7 – просте число і 9 – просте число;

в) 7 – просте число або 9 – просте число;

г) число 2 парне або це число просте;

д) $2 \cdot 2 = 4$ або білі ведмеді живуть в Африці;

е) $2 \cdot 2 = 4$ та $2 \cdot 2 < 5$ та $2 \cdot 2 > 4$;

ж) $3 \cdot 3 = 9$ та $4 + 7 = 11$.

4. Визначити значення істинності висловів *A, B, C, D, E, F, G, H*:

а) $A \,\&\, (2 \cdot 2 = 4) = 1$; д) $\neg E \vee (2 \cdot 2 = 5) = 1$;

б) $B \vee (2 \cdot 2 = 5) = 1$; е) $F \,\&\, (2 \cdot 2 = 4) = 0$;

в) $C \vee (2 \cdot 2 = 4) = 1$; ж) $G \vee (2 \cdot 2 = 5) = 0$;

г) $\neg D \,\&\, (2 \cdot 2 = 4) = 1$; з) $H \,\&\, (2 \cdot 2 = 5) = 0$.

5. Визначити значення істинності таких висловів:

а) якщо 9 ділиться на 3, то 4 ділиться на 2;

б) якщо 11 ділиться на 6, то 11 ділиться на 3;

в) 12 ділиться на 6 тоді і тільки тоді, коли 12 ділиться на 3;

г) $4 > 5$ тоді і тільки тоді, коли $-4 > -5$;

д) 15 ділиться на 5 тоді і тільки тоді, коли 15 ділиться на 4;

е) 11 ділиться на 6 тоді і тільки тоді, коли 11 ділиться на 3.

6. Визначити значення істинності висловів *A*, *B*, *C*, *D*:

а) $A \leftrightarrow (2 < 3) = 1$;

б) $B \leftrightarrow (2 > 3) = 1$;

в) $(6 < 7) \leftrightarrow \neg C = 1$;

г) $(6 > 7) \leftrightarrow \neg D = 1$.

7. Записати такі положення за допомогою формул:

а) вважаємо, що команда виграла турнір (*A*), якщо забито 10 голів (*B*), не було програшів (*C*) і не було порушень (*D*);

б) потрібно отримати зарплату (*A*), для того, щоб діти отримали подарунки (*C*) або пішли на новорічне свято (*B*);

в) якщо вологість така висока (*A*), то або пополудні (*B*), або ввечері (*C*) піде дощ;

г) задача має бути сформульованою (*B*) і мати рішення (*C*), для того, щоб її можна було вирішити (*A*);

д) якщо студент складе заліки (*A*), лабораторні роботи (*B*) і курсові (*C*), то його допустять до складання іспитів (*D*).

8. Нехай є вислови

P := «йому потрібний лікар»; *Q* := «йому потрібний адвокат»;

R := «з ним стався нещасний випадок»; *S* := «він хворий»; *U* := «він поранений».

Записати такі положення за допомогою формул:

а) якщо він хворий, йому потрібний лікар, і якщо з ним стався нещасний випадок, йому потрібний адвокат;

б) якщо йому потрібний лікар і адвокат, то стався нещасний випадок;

в) якщо йому потрібний лікар, то він хворий або поранений;

г) потрібний лікар і адвокат лише за умови, що він хворий або поранений;

д) якщо він не хворий або не поранений, то йому не потрібний лікар.

9. Заповнити істиннісні таблиці для формул:

а) $G = (P \vee Q) \leftrightarrow (Q \rightarrow (\neg P))$;

б) $G = (\neg P \,\&\, \neg Q) \rightarrow (P \vee Q)$;

в) $G = (\neg P \vee Q) \leftrightarrow (\neg (P \,\&\, Q))$;

г) $G = (P \rightarrow \neg Q) \,\&\, (\neg P \vee Q)$;

д) $G = (P \,\&\, \neg Q) \vee (\neg P \leftrightarrow Q)$.

10. Визначити для формули властивість загальнозначущості або суперечності:

а) $\neg P \,\&\, (\neg (P \rightarrow Q))$;

б) $(P \rightarrow Q) \rightarrow (\neg Q \rightarrow \neg P)$;

в) $(P \,\&\, \neg Q) \,\&\, (\neg P \vee Q)$;

г) $(P \,\&\, (Q \rightarrow P)) \rightarrow P$;

д) $(\neg Q \leftrightarrow P) \,\&\, (\neg (P \vee Q))$.

11. Перевірити еквівалентність таких формул, перетворюючи формули з обох боків від знака $\cong$ до однієї і тієї ж нормальної форми:

а) $(P \rightarrow Q) \,\&\, (P \rightarrow R) \cong (P \rightarrow (Q \,\&\, R))$;

б) $(P \rightarrow Q) \rightarrow (P \,\&\, Q) \cong (\neg P \rightarrow Q) \,\&\, (Q \rightarrow P)$;

в) $P \,\&\, Q \,\&\, (\neg P \vee \neg Q) \cong \neg P \,\&\, \neg Q \,\&\, (P \vee Q)$.

12. Довести еквівалентність таких формул:

а) $(A \vee B) \,\&\, (A \vee \neg B) \cong A$;

б) $(A \vee B) \,\&\, (B \vee C) \,\&\, (C \vee A) \cong (A \,\&\, B) \vee (B \,\&\, C) \vee (C \,\&\, A)$;

в) $(A \vee B) \,\&\, (A \vee C) \,\&\, (B \vee D) \,\&\, (C \vee D) \cong ((A \,\&\, D) \vee (B \,\&\, C))$.

13. Один з трьох братів поставив на скатертину пляму.

– Вітя не ставив пляму, – сказав Олексій. – Це зробив Борис.

– Це Вітя поставив пляму, – сказав Борис. – А Олексій не бруднив скатертину.

– Я знаю, що Борис не міг це зробити. А я сьогодні не готував уроки, – сказав Вітя.

Виявилось, що двоє хлопчиків в кожному з двох випадків сказали правду, а один обидва рази сказав неправду. Хто поставив на скатертину пляму?

Вказівка. Утворюйте перш за все всілякі попарні диз'юнкції з висловів братів. Всі вони будуть істинні. Потім розглянете кон'юнкцію всіх дійсних диз'юнкцій. Вона теж буде істинна. Перетворивши її до кон'юнкції елементарних висловів, встановите винного.

14. Один з чотирьох хлопчиків зіпсував вимикач. На запитання «Хто це зробив?» були отримані такі відповіді:

1) «Це зробив або Міша, або Коля»;

2) «Це зробив або Вітя, або Коля»;

3) «Це не могли зробити ні Міша, ні Толя»;

4) «Це зробив або Вітя, або Міша».

Чи можна за цими даними встановити, хто винен у поломці вимикача, якщо з чотирьох висловів три – істинні?

15. Нехай $P(x)$ означає вислів «x – раціональне число» і $Q(x)$ – «x – дійсне число». Записати символами такі речення:

а) кожне раціональне число є дійсне число;

б) деяке дійсне число є раціональне число;

в) не кожне дійсне число є раціональне число.

16. Нехай $C(x)$ означає «x – торговець старими автомобілями» і $H(x)$ означає «x – чесна людина». Перекласти такі вирази українською мовою:

а) $(\exists x)\ C(x)$;

б) $(\exists x)\ H(x)$;

в) $(\forall x)(C(x) \to \neg H(x))$;

г) $(\exists x)\ (C\ (x)\ \&\ H\ (x))$;

д) $(\exists x)(H(x) \to C(x))$.

17. Для цієї інтерпретації $D = \{a,\ b\}$,

$P(a,a)$	$P(a,b)$	$P(b,a)$	$P(b,b)$
1	0	1	0

визначити істиннісні значення таких формул:

а) $(\forall x)(\exists y)\ P(x, y)$;

б) $(\forall x)(\forall y)\ P(x, y)$;

в) $(\exists x)(\forall y)\ P(x, y)$;

г) $(\exists y) \neg P(a, y)$;

д) $(\forall x)(\forall y)\ (P\ (x, y) \leftrightarrow P(y, x))$;

е) $(\forall x)P(x, x)$.

18. Розглянемо таку формулу:

$$(\exists x)\ P(x) \to (\forall x)\ \mathrm{P}(x).$$

а) Довести, що ця формула завжди істинна, якщо область D містить тільки один елемент.

б) Нехай $D = \{a, b\}$. Знайти інтерпретацію з областю D, в якій a має значення *хибність*.

19. Дана така інтерпретація:

Область: $D = \{1, 2\}$.

Значення констант a і b: $a = 1$; $b = 2$.

Значення функції f: $f(1) = 2$; $f(2) = 1$.

Значення предиката P:

$P(1,1)$	$P(1,2)$	$P(2,1)$	$P(2,2)$
1	1	0	0

Знайти істиннісні значення цих формул у вказаній інтерпретації:

а) $P(a, f(a)) \,\&\, P(b, f(b))$;

б) $(\forall x)(\exists y)\, P\,(y, x)$;

в) $(\forall x)(\forall y)\,(P(x, y) \to P\,(f\,(x), f\,(y)))$.

20. Перетворити такі формули в передуючу нормальну форму:

а) $(\forall x)\,(P\,(x) \to (\exists y)\, Q(x, y))$;

б) $(\exists x)\,(\neg((\exists y)\, P\,(x, y)) \to ((\exists z)\, Q\,(z) \to R\,(x)))$;

в) $(\forall x)(\forall y)((\exists z)\, P\,(x, y, z)\, \&\, ((\exists u)\, Q\,(x, u) \to (\exists v)\, Q\,(y, v)))$.

21. Привести до передуючої нормальної форми:

а) $\exists x(\forall y(P_1(x; y)))\&\exists x(\forall y(P_2(x; y)))$;

б) $\exists x(\forall y(P_1(x; y)))\vee\exists x(\forall y(P_2(x; y)))$;

в) $\exists x(\forall y(P_1(x; y)))\to\exists x(\forall y(P_2(x; y)))$;

г) $\forall x(\forall y(P_1(x; y)))\to\exists x(\exists y(P_2(x; y)))$;

д) $\forall y(P_1(x)\vee(P_2(x))\to\forall y(P_1(x))\vee\forall y\,(P_2(x))$.

22. Визначити істинність імплікації за відомими значеннями істинності засновку A і висновків B і C.

а) «Якщо він сьогодні промочить ноги, то завтра він захворіє». Для вислову A = «він сьогодні промочить ноги» прийнято $\rho(A) = 0{,}7$, для вислову B = «завтра він захворіє» прийнято $\rho(B) = 0{,}9$.

б) «Якщо добре вивчити урок, то можна отримати відмінну оцінку, інакше можна отримати двійку». Для вислову A = «добре вивчити урок» прийнято $\rho(A) = 0{,}5$, для вислову B = «можна отримати відмінну оцінку» – $\rho(B) = 0{,}8$, для вислову C = «можна отримати двійку» – $\rho(C) = 0{,}1$.

23. Нехай $U = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8\}$. Виконати операції об'єднання, перетину, доповнення, різниці і симетричної різниці над нечіткими множинами A і B.

а) $A = \{\ 1/u_1\ ,\ 0{,}1/u_2\ ,\ 0{,}2/u_3\ ,\ 0{,}3/u_4\ ,\ 0{,}4/u_5\ \}$;
$B = \{0{,}1/u_1,\ 0{,}2/u_2,\ 0{,}3/u_6,\ 0{,}6/u_7,\ 0{,}8/u_8\}$.

б) $A = \{\ 0{,}1/u_1\ ,\ 0{,}7/u_3\ ,\ 0{,}8/u_6\ ,\ 1{,}0/u_7\ ,\ 0{,}5/u_8\ \}$;
$B = \{1{,}0/u_2,\ 0{,}3/u_3,\ 0{,}7/u_4,\ 0{,}7/u_5,\ 0{,}4/u_8\}$.

в) $A = \{\ 0{,}1/u_1\ ,\ 0{,}1/u_3\ ,\ 0{,}1/u_5\ ,\ 0{,}1/u_7\}$;
$B = \{1{,}0/u_2,\ 1{,}0/u_4,\ 1{,}0/u_6,\ 1{,}0/u_8\}$.

3. БУЛЕВІ ФУНКЦІЇ. ОСНОВНІ ЗАКОНИ АЛГЕБРИ ЛОГІКИ

3.1. Цифрові автомати в схемотехніці та програмуванні

Оточуюча нас інформація постійно зазнає перетворення. Але не всі оточуючі нас перетворювачі інформації виконують тільки функціональне відображення інформації вхід – вихід. Результат перетворення часто залежить не тільки від того, яка інформація з'явилась на вході, але і від того, що відбувалось раніше.

Наприклад, один і той же вхід – вибачення сусіда після того, як він наступив вам на ногу, викличе у вас одну реакцію першого разу і зовсім іншу – на п'ятий раз.

Автомат – це приклад пристрою, реакція якого залежить не тільки від входу, але і від того, що було раніше, тобто від стану в попередній момент часу.

Комп'ютер – приклад цифрового автомата. Він має пам'ять, яка зберігає його стан. Залежно від стану комп'ютер виконує ту чи іншу дію. Синонім терміна «цифровий автомат» – «скінченний автомат». Перший термін підкреслює, що автомат працює з цифрами, тобто з наборами символів, другий – що його пам'ять скінченна.

Але комп'ютер досить складний пристрій для тих методів, які будуть розглянуті далі. Більш підходящий приклад – команди в операційній системі *Unix*, кожна з яких є скінченним автоматом.

Скінченний автомат – абстракція, яка дозволяє не розглядати динамічні стани, які виникають під час перехідних процесів. Скінченний автомат враховує стани перед початком і після завершення переходу. В проміжні моменти часу скінченно-автоматний опис неможливий. Методи конструювання реальних пристроїв дозволяють розглядати скінченно-автоматний опис як опис функції автомата, а перехідні процеси врахувати заздалегідь і сховати їх від зовнішнього відображення. Якщо ця задача вирішена, то можна розглядати пристрій як скінченний автомат. Ще більш просте співвідношення між скінченним автоматом і програмою. Будь-який скінченний автомат може бути представлений програмою.

Далі використовується термін «цифровий автомат» для пристроїв, які реалізують дії над числами. Це – суматори, пристрої для множення та ділення, пристрої для добування кореня та обчислення значень тригонометричних функцій і для виконання інших операцій, які розробник вважає за необхідне реалізувати апаратно.

Відмінності цифрового автомата від скінченного автомата загального виду полягають у такому:

- він призначений для подання чисел і виконання операцій над ними;
- у ньому на самому першому етапі, етапі синтезу однорозрядного або багаторозрядного суматора, вирішується проблема гонок (породжуваних затримками);
- цифровий автомат має похибку подання – похибку, яка виникає при представленні лічильних або нелічильних числових систем через обмежену

кількість розрядів у комірках пам’яті. Відмінності реалізованої операції від арифметичної в особливо важливих випадках відображаються додатковими сигналами – переповненням або сигналом «машинного нуля»;

- цифровий автомат занадто складний для автоматичного синтезу.

Апаратно реалізуються в основному суматори, а також пристрої для множення і ділення. Більш складні операції зводяться до вказаних, так що арифметичний пристрій використовується одночасно з управляючим автоматом, який реалізується або апаратно, або програмно.

3.2. Висловлювання, предикати, булеві функції

ВИСЛОВЛЮВАННЯМ називається розповідне речення, яке може бути істинним або хибним. Якщо про будь-яку частину речення не можна сказати, вона істинна чи хибна, то говорять, що ВИСЛОВЛЮВАННЯ ПРОСТЕ. Наприклад, «сьогодні гарна погода». Це висловлювання просте.

Звичайно в реченні можна виділити частини, які самі являються висловлюваннями, що з’єднані сполучниками «і», «або», «інакше», «якщо…то…» і т. ін. Але розбиття складного розповідного речення на прості висловлювання треба робити за смислом, а не за словами. В реченні *«у цій аудиторії присутні студенти і викладач»* частинами-висловлюваннями являються (*у цій аудиторії присутні студенти*), (*у цій аудиторії присутній викладач*), які з’єднані сполучником *і*.

Суть сполучника *і* така, що значенням складного висловлювання буде ***істина***, якщо істинні обидва прості висловлювання, які входять в нього, і буде ***хибність***, якщо хоча б одне просте висловлювання хибне. Неправильна форма: (*у цій аудиторії присутні студенти*) *і* (*викладач*). Тут зміст других дужок не є висловлюванням, оскільки він не може бути істинним або хибним.

Правил перетворення складного речення в набір простих висловлювань, з’єднаних сполучниками, не існує. Ця трудність притаманна всім природним мовам, і вона дуже ускладнює, зокрема, переклад з однієї мови на іншу. Однак таке перетворення в більшості випадків легко виконує будь-хто, хто володіє необхідними мовами. *Правило*: в складному реченні сполучники треба використовувати тільки між висловлюваннями.

Якщо висловлювання містить змінні, то воно називається ЛОГІЧНИМ ВИРАЗОМ або **ПРЕДИКАТОМ.** Наприклад, (x більше 0). Логічний вираз (предикат) може бути істинним або хибним залежно від значення змінної x. Іншими словами, предикат – функція від змінних, яка приймає тільки два значення – {***істина, хибність***}. Для предикатів використовують функціональне позначення: $M(x) = (x > 0)$.

Предикати дозволяють формалізувати ВЛАСТИВОСТІ ЗМІННИХ:
$M(x) =$ ***true*** означає, що x володіє властивістю ($x > 0$), а $M(x) =$ ***false*** означає, що x не володіє властивістю ($x > 0$).

На алгоритмічній мові можна написати програму для обчислення значень вказаного предиката:

```
function M(x: real):boolean;          int M(float x)
begin                          або    { if (x>0) return 1;  else return ; }
   M:= (x>0)
end;
```

БУЛЕВОЮ ЗМІННОЮ називається змінна, яка приймає значення **{*істина, хибність*}**.

БУЛЕВОЮ ФУНКЦІЄЮ називають предикат, у якого змінні є булевими.

Такою функцією, наприклад, є складне висловлювання (*в аудиторії присутні студенти*) *і* (*в аудиторії присутній викладач*), в якому перша і друга дужки розглядаються як змінні.

Можна також сказати, що булева функція – це функція від змінних, які приймають два значення, і множина її значень складається з двох елементів. Звичайно ці два значення позначаються або {**0, 1**}, або {***false, true***}. Найчастіше позначення {**0**, **1**} вибирають, коли неможливо сплутати логічне значення *істина* з числом 1. Якщо ж у тексті використовуються як числа, так і логічні значення, то вибирають друге позначення.

В алгоритмічній мові Паскаль є дві логічні константи ***false*** і ***true*** та тип *Boolean* = {***false, true***}, за допомогою якого можна вводити логічні змінні:

var x, y: *Boolean*;

Сполучники ***І, АБО, НІ, АБО…АБО…, ЯКЩО…, ТО…*** використовуються в логіці.

Суть цих сполучників така:

- *A* ***І*** *B* – істинно, якщо істинні і *A,* і *B*;
- *A* ***АБО*** *B* – істинно, якщо істинне *A*, або істинне *B*, або істинні обидва;
- ***НІ*** *A* – істинно, якщо *A* хибне;
- ***АБО*** *A*, ***АБО*** *B* – істинно, якщо істинне *A* або *B*, але не обидва зразу;
- ***ЯКЩО*** *A*, ***ТО*** *B* – хибно, якщо істинне *A* і хибне *B*. У всіх інших випадках це істинно. Наприклад, ***ЯКЩО*** хибність, ***ТО*** *B* – істинно незалежно від значення *B*. Говорять у зв’язку з цим, що із хибності слідує що завгодно.

Сполучникам ***І, АБО, НІ, АБО…АБО…*** в алгоритмічній мові Паскаль відповідають булеві оператори: ***and, or, not, xor*** – «і», «або», «ні», «розподілювальне або». Оператор ***and*** називається також КОН’ЮНКЦІЄЮ, оператор ***or*** – ДИЗ’ЮНКЦІЄЮ, оператор ***not*** – ЗАПЕРЕЧЕННЯМ. Вирази виду (***x or y***), (***x and y***), ((***x and not*** (***y***)) ***or*** (***not*** (***x***) ***and y***) і т. ін. – називаються булевими.

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

У булевих виразах використовуються оператори: ***and, or, not, xor*** – «І», «АБО», «НІ», «виключне АБО». Кожний булевий вираз задає булеву функцію.

Але правильно і те, що кожна булева функція може бути задана булевим виразом.

Найбільш поширеними способами подання булевих функцій є: табличний, метод перелічення, цифровий еквівалент, аналітичний, семантичне дерево.

При використанні **табличного способу** булева функція $f(x_1, \ldots, x_n)$ задається таблицею істинності (табл. 3.1), у всіх стовпцях якої, крім крайнього правого, представлені всі можливі двійкові набори довжини n, а в крайньому правому стовпці вказуються значення функції на цих наборах.

Таблиця 3.1 – Таблиця істинності функції f_1

x_1	x_2	x_3	f_1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Під двійковим набором $\gamma = \langle \gamma_1, \gamma_2, \ldots \gamma_n \rangle, \gamma_i \in \{0,1\}$ розуміється сукупність значень аргументів $x_1, x_2, \ldots, x_n$ булевої функції f. Двійковий набір має довжину n, якщо він поданий n елементами з множини {**0**, **1**}. У табл. 3.1 перелічені всі двійкові набори довжини три відповідно.

Іноді двійкові набори в таблиці істинності булевої функції зручно представляти номерами наборів. Запишемо аргументи $x_1, x_2, \ldots, x_n$ в порядку збільшення їх індексів. Тоді будь-який двійковий набір $\gamma = \langle \gamma_1, \gamma_2, \ldots \gamma_n \rangle, \gamma_i \in \{0,1\}$ можна розглядати як ціле двійкове число N

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

Воно називається номером набору. Наприклад, двійкові набори 0101 і 1000 мають номери 5 і 8 відповідно. Очевидно, що будь-яка булева функція може бути задана **переліченням наборів**, які приймають, наприклад, одиничне значення. Функція f_1 в цьому випадку може бути представлена таким чином: $f_1(x_1, x_2, x_3) = (2, 3, 5, 7)^1$.

Булеву функцію, задану на всіх її наборах, називають **повністю визначеною.** В табл. 3.1 наведений приклад повністю визначеної булевої функції.

Булеву функцію n змінних називають **не повністю визначеною** або **частково визначеною**, якщо вона задана не на всіх двійкових наборах довжини n. Не повністю визначена булева функція не підпадає під визначення, наведене вище, але при довільному довизначенні (на всіх наборах, на яких вона не визначена) ця невідповідність знімається.

Легко переконатись, що якщо булева функція f_2 не визначена на $m = 3$ наборах аргументів, то шляхом її довизначення можна отримати 2^m різних повністю визначених функцій. В табл. 3.2 наведений приклад не повністю визначеної булевої функції чотирьох змінних.

Таблиця 3.2 – Таблиця істинності функції f_2

x_1	x_2	x_3	x_4	f_2
0	0	0	0	0
0	0	0	1	0
0	0	1	0	*
0	0	1	1	1
0	1	0	0	0
0	1	0	1	*
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	*
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Цифровий еквівалент подання функції можна отримати, якщо виписати усі значення функції, починаючи зі старшого номера двійкових наборів, а потім отриману двійкову послідовність перевести в десяткове число. Наприклад, в табл. 3.1 значення $f_1(x_1, x_2, x_3)$ являють собою послідовність 10101100_2 => $128 + 32 + 8 + 4 = 172_{10}$.

Аналітичне подання булевої функції являє собою логічну формулу. Наприклад, аналітичне подання функції:

$$f_1(x_1, x_2, x_3) = \overline{x_1}x_2 \vee x_1x_3.$$

Семантичне дерево – це двійкове дерево, корінь якого помічений двійковою функцією від n змінних. Із кожного вузла виходять по два ребра, що відповідають двом значенням (нуль і одиниця) наступної змінної, а 2^n листочків помічені відповідними значеннями двійкової функції

3.3. Схемні реалізації булевих функцій

Булеві функції використовуються під час синтезу пристроїв управління. Кожна булева функція з базису *and, or, not* має реалізацію у вигляді мікросхеми (рис. 3.1). Заперечення зображається колом на вході або на виході. Символ функції вписується в рамку елемента. Як символ операції *or* використовується символ «1», а операції *and* – символ «&».

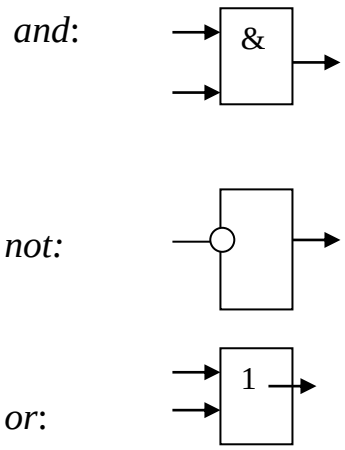


Рисунок 3.1 – Реалізація булевих функцій

Ці елементи об’єднуються в схему відповідно до аналітичного виразу для функції.

Наприклад, $f_3 = \overline{x_1} x_3 x_4 \vee \overline{x_1} x_2 x_4 \vee x_2 x_3 x_4$ реалізується такою схемою (рис. 3.2).

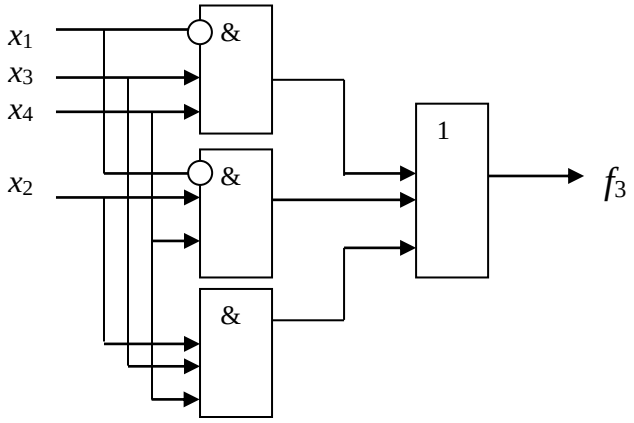


Рисунок 3.2 – Реалізація функції f_3

3.4. Найбільш поширені булеві функції

Існує не більше ніж 2^{2^n} різних булевих функцій n змінних. До цього висновку легко дійти, користуючись простими комбінаторними міркуваннями і згадавши, що на кожному з 2^n наборів функції можуть приймати два значення.

Розглянемо найбільш використовувані булеві функції однієї і двох змінних.

Функції однієї змінної подані в табл. 3.3, де:

$f_0(x) = 0$ – тотожний нуль (константа 0);

$f_1(x) = x$ – тотожна функція x;

$f_2(x) = \overline{x}$ – заперечення x (інверсія x);

$f_3(x) = 1$ – тотожна одиниця (константа 1).

Таблиця 3.3 – Функції однієї змінної

x	f_0	f_1	f_2	f_3
0	0	0	1	1
1	0	1	0	1

Функції двох змінних подані в табл. 3.4.

Таблиця 3.4 – Функції 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	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

Найбільш часто використовуються такі найменування цих функцій:

$f_0(x_1, x_2) = 0$ – тотожний нуль (константа 0);

$f_1(x_1, x_2) = x_1 \bullet x_2$ – кон'юнкція (логічне ***І***);

$f_2(x_1, x_2) = x_1 \bullet \bar{x}_2 = x_2 \leftarrow x_1$ – заперечення x_1;

$f_3(x_1, x_2) = x_1$ – повторення x_1;

$f_4(x_1, x_2) = \bar{x}_1 \bullet x_2 = x_1 \leftarrow x_2$ – заперечення x_2;

$f_5(x_1, x_2) = x_2$ – повторення x_2;

$f_6(x_1, x_2) = x_1 \oplus x_2$ – виключне АБО;

$f_7(x_1, x_2) = x_1 \vee x_2$ – диз'юнкція (логічне ***АБО***);

$f_8(x_1, x_2) = x_1 \downarrow x_2$ – стрілка Пірса (заперечення кон'юнкції, операція ***АБО-НІ***);

$f_9(x_1, x_2) = x_1 \sim x_2$ – еквівалентність;

$f_{10}(x_1, x_2) = \bar{x}_2$ – повторення $\bar{x}_2$;

$f_{11}(x_1, x_2) = x_2 \rightarrow x_1$ – імплікація;

$f_{12}(x_1, x_2) = \bar{x}_1$ – повторення $\bar{x}_1$;

$f_{13}(x_1, x_2) = x_1 \rightarrow x_2$ – імплікація;

$f_{14}(x_1, x_2) = x_1 / x_2$ – штрих Шеффера (заперечення диз'юнкції, операція ***І-НІ***);

$f_{15}(x_1, x_2) = 1$ – тотожна одиниця (константа 1).

Функцію $f_1(x_1, x_2)$ часто називають логічним множенням. Тут замість знака «•» використовують також знак «&» або «$\wedge$».

Очевидно, що серед схем, які реалізують дану функцію, є найбільш проста. Пошук логічної формули, що відповідає цій схемі, має великий практичний інтерес, а перетворення формул булевих функцій основано на використанні співвідношень булевої алгебри.

Для булевої алгебри визначені одна одномісна (унарна) операція – «заперечення» і дві двомісні (бінарні) операції – «кон'юнкція» та «диз'юнкція» (позначаються символами «•», «$\vee$» відповідно).

3.5. Основні закони алгебри логіки

Під бінарною операцією на множині A в загальному випадку розуміють відображення декартового добутку множин ($A \times A$) в множину A. Іншими словами, результат використання бінарної операції до будь-якої упорядкованої пари елементів з A є також елемент із множини A.

Під унарною операцією на множині A розуміють виділення (фіксацію) якогось елемента множини A.

У булевій алгебрі справедливі закони:

закон комутативності – $x \vee y = y \vee x, x \cdot y = y \cdot x;$

закон асоціативності – $(x \vee y) \vee z = x \vee (y \vee z), (x \cdot y) \cdot z = x \cdot (y \cdot z);$

закон дистрибутивності – $x \cdot (y \vee z) = x \cdot y \vee x \cdot z, x \vee y \cdot z = (x \vee y) \cdot (x \vee z);$

закон повторення – $x \vee x = x, x \cdot x = x, \overline{\overline{x}} = x;$

закон універсальності меж – $x \vee \overline{x} = 1, x \cdot \overline{x} = 0;$

закон інволютивності – $\overline{\overline{x}} = x;$

закон доповнення – $x \vee 1 = 1, x \cdot 0 = 0;$

закон поглинання – $x \vee x \cdot y = x, x \cdot (x \vee y) = x;$

закон спрощення – $x \vee \overline{x} y = x \vee y;$

закон склеювання – $x \cdot y \vee x \cdot \overline{y} = x, (x \vee y) \cdot (x \vee \overline{y}) = x;$

закон де Моргана – $\overline{x \vee y} = \overline{x} \cdot \overline{y}, \overline{x \cdot y} = \overline{x} \vee \overline{y}.$

Використовуючи дані залежності, можна перетворити початкові вирази в більш прості, тобто мінімізувати їх. За спрощеними виразами можна побудувати технічний пристрій з мінімальними апаратурними витратами.

Приклад 1. Спростити вираз $y = x_1 \cdot x_2 \cdot x_3 \vee x_1 \cdot \overline{x_2} \cdot x_3 \vee x_1 \cdot x_3$.

Використовуючи закон склеювання, отримаємо

$$y = x_1 \cdot x_3 \cdot (x_2 \vee \overline{x_2}) \vee x_1 \cdot x_3 = x_1 \cdot x_3 \vee x_1 \cdot x_3 = x_1 \cdot x_3.$$

Приклад 2. Спростити вираз $N = \overline{\overline{(x \cdot \overline{y} \vee \overline{x \cdot y}) \cdot x} \vee (x \cdot \overline{y} \vee \overline{x} \cdot y) \cdot x}$.

Використовуючи закон де Моргана, отримаємо

$$N = \overline{[(x \cdot \overline{y} \vee \overline{x} \cdot y) \cdot x]} \cdot \overline{[(x \cdot \overline{y} \vee \overline{x} \cdot y) \cdot x]}.$$

Використовуючи закон де Моргана, маємо

$$N = \overline{\overline{[(x \cdot \overline{y} \vee \overline{x} \cdot y) \vee \overline{x}]}} \cdot [\overline{(x \cdot \overline{y} \vee \overline{x} \cdot y)} \vee \overline{x}].$$

Використовуючи закон ідемпотентності до першої круглої дужки і закон де Моргана до другої круглої дужки, отримаємо

$$N = (x \cdot \overline{y} \vee \overline{x} \cdot y \vee \overline{x}) \cdot [\overline{\overline{x \cdot y}} \cdot \overline{\overline{x} \cdot y}) \vee \overline{x}].$$

Послідовно застосовуючи закон поглинання та закон спрощення, а до виразів із запереченнями – закон де Моргана, отримаємо

$$N = (\bar{y} \vee \bar{x}) \cdot [(\bar{x} \vee y) \cdot (x \vee \bar{y}) \vee \bar{x}].$$

Після перетворення маємо

$$N = (\bar{y} \vee \bar{x}) \cdot (\bar{x} \cdot \bar{y} \vee x \cdot y) \vee \bar{x}).$$

І нарешті, використовуючи закон поглинання та закон спрощення, отримаємо

$$N = (\bar{y} \vee \bar{x}) \cdot (\bar{x} \vee y) = \bar{y} \cdot \bar{x} \vee \bar{x} \cdot y = \bar{x}.$$

Контрольні запитання

1. Що означають терміни «цифровий автомат» і «скінченний автомат»?
2. Дайте пояснення терміна «висловлювання».
3. Яке висловлювання називається предикатом?
4. Які змінні та функції являються булевими?
5. Які булеві оператори використовуються в булевих виразах?
6. Охарактеризуйте різні способи подання булевих функцій.
7. Наведіть приклад схемного відтворення булевої функції.
8. Перелічіть найменування булевих функцій однієї і двох змінних.
9. Які основні закони алгебри логіки ви знаєте?
10. Продемонструйте використання законів алгебри логіки для
11. спрощення логічних виразів.
12. Виразити функції АБО, І та НІ через функції АБО-НІ.
13. Спростити вираз

$$N = \overline{x_1} \vee \overline{x_2} x_1 \vee 0 \vee 1.$$

14. Виразити функції АБО, І та НІ через функції І-НІ.
15. Побудуйте таблицю істинності для функції $f_1(x_1, x_2, x_3) = (0,3,4,6,7)^1$.
16. Спростити вираз $N = \overline{x/x}/1/0/x/\bar{x}$.
17. Спростити вираз $N = x \downarrow x \downarrow 1 \downarrow 0 \downarrow x \downarrow \bar{x}$.

4. АНАЛІТИЧНЕ ПОДАННЯ БУЛЕВИХ ФУНКЦІЙ. ФУНКЦІОНАЛЬНО ПОВНІ СИСТЕМИ БУЛЕВИХ ФУНКЦІЙ

Розглянуті найпростіші булеві функції дозволяють будувати нові булеві функції за допомогою узагальненої операції, яка називається операцією **суперпозиції**. Фактично операція суперпозиції полягає в підстановці замість аргументів даної функції інших булевих функцій (зокрема інших аргументів). Наприклад, із функції $f_1(x_1, x_2)$ за допомогою підстановок $x_1 = f_3(x_5, x_4)$ і $x_2 = x_3$ замість аргументів x_1 і x_2 відповідно отримаємо функцію $f_1(f_3(x_5, x_4), x_3)$. Остання від змінних x_1 і x_2 уже не залежить.

Операція суперпозиції дозволяє побачити якісний перехід від n, рівного одиниці, до n, рівного двом. Дійсно, суперпозиція функцій одного аргументу породжує функції також одного аргументу. Суперпозиція функцій двох аргументів дає можливість будувати функції будь-якої кількості аргументів (у наведеному прикладі побудована функція трьох аргументів).

Суперпозиція булевих функцій зображається у вигляді логічних формул. Однак необхідно відмітити:

- одна і та ж функція може бути представлена різними формулами;
- кожній формулі відповідає своя суперпозиція і, отже, своя схема з'єднання логічних елементів;
- між формулами подання булевих функцій і схемами, які їх реалізують, існує взаємно однозначна відповідність.

Будь-яка двійкова функція, за винятком функції тотожного нуля, може бути подана як суперпозиція трьох функцій ***І, АБО*** та ***НІ***.

4.1. Досконала диз'юнктивна нормальна форма

Оскільки між множиною аналітичного подавання і множиною схем, які реалізують булеву функцію, існує взаємно однозначна відповідність, відшукування канонічної форми подання булевої функції є початковим етапом синтезу схеми, що її реалізує. Найбільше поширення отримала **досконала диз'юнктивна нормальна форма** (ДДНФ). Наведемо визначення конституенти одиниці – поняття, яке буде далі широко використовуватись.

Визначення. Конституентою одиниці називається функція $f(x_1, x_2, \ldots, x_n)$, яка приймає значення 1 тільки на одному наборі.

Конституента одиниці записується у вигляді **терма**, тобто у вигляді логічного множення n різних булевих змінних, деякі з яких можуть бути з використанням заперечення. Наприклад, $f = x_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \cdot x_4$ – елементарний логічний добуток, який являється конституентою одиниці змінних x_1, x_2, x_3 і x_4, приймає значення 1 тільки на одному наборі 1001. Зрозуміло, що на решті 15 наборах ця функція приймає значення 0.

Якщо згадати, що диз'юнкція приймає значення 1, коли хоча б одна зі змінних приймає значення 1, то можна легко виразити будь-яку булеву функцію як диз'юнкцію конституент одиниці, які відповідають тим наборам, на

яких функція приймає значення 1. У більш загальному вигляді це можна записати таким чином (4.1):

$$f(x_1, x_2, ..., x_n) = \bigvee_{\sigma} f(\sigma_1, \sigma_2, ..., \sigma_n) \cdot x_1^{\sigma_1} \cdot x_2^{\sigma_2} \cdot ... \cdot x_n^{\sigma_n},$$

$$\text{де } \sigma_i = 0,1 \quad \text{і} \quad x_i^{\sigma_{i1}} = \begin{cases} x \sim \sigma_i = 1 \\ \bar{x} \sim \sigma_i = 0 \end{cases}. \qquad (4.1)$$

Ця форма і є ДДНФ. Відмітимо, що набори, на яких функція *f* приймає значення 1, часто називаються одиничними, а інші – нульовими наборами. Виписувати в ДДНФ доцільно тільки конституенти одиниці, тобто те, що відповідає одиничним наборам. Подання булевої функції в ДДНФ єдине.

Таблиця 4.1 – Таблиця істинності

x_1	x_2	x_3	f_1	f_2
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

Приклад 1. Випишемо ДДНФ для функцій, заданих таблицею істинності (табл. 4.1).

$$f_1 = 0 \cdot \bar{x}_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \vee 1 \cdot \bar{x}_1 \cdot \bar{x}_2 \cdot x_3 \vee 1 \cdot \bar{x}_1 \cdot x_2 \cdot \bar{x}_3 \vee$$
$$\vee 0 \cdot \bar{x}_1 \cdot x_2 \cdot x_3 \vee 1 \cdot x_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \vee 0 \cdot x_1 \cdot \bar{x}_2 \cdot x_3 \vee$$
$$\vee 0 \cdot x_1 \cdot x_2 \cdot \bar{x}_3 \vee 1 \cdot x_1 \cdot x_2 \cdot x_3 =$$
$$= \bar{x}_1 \bar{x}_2 x_3 \vee \bar{x}_1 x_2 \bar{x}_3 \vee x_1 \bar{x}_2 \bar{x}_3 \vee x_1 x_2 x_3;$$

$$f_2 = \bar{x}_1 x_2 x_3 \vee x_1 \bar{x}_2 x_3 \vee x_1 x_2 \bar{x}_3 \vee x_1 x_2 x_3.$$

4.2. Досконала кон'юнктивна нормальна форма

Друга відома форма носить назву **досконалої кон'юнктивної нормальної форми** (ДКНФ). Вона будується аналогічно ДДНФ, але при цьому використовуються конституенти нуля.

Визначення. Конституентою нуля називається функція, яка приймає значення 0 тільки на одному наборі.

Конституента нуля записується у вигляді елементарної диз'юнкції всіх змінних даної функції. Кожному набору відповідає своя конституента нуля. Наприклад, набору 0110 змінних x_1, x_2, x_3 і x_4 відповідає конституента нуля $x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4$. ДКНФ представляється як кон'юнкція конституент нуля, які відповідають тим наборам, на яких функція приймає значення 0. У більш загальному вигляді це можна записати таким чином (4.2):

$$f(x_1, x_2, ..., x_n) = \mathop{.}_{\sigma} f(\sigma_1, \sigma_2, ..., \sigma_n) \cdot x_1^{\sigma_1} \vee \cdot x_2^{\sigma_2} \vee ... \vee x_n^{\sigma_n},$$

$$\text{де } \sigma_i = 0,1 \quad \text{і} \quad x_i^{\sigma_{i1}} = \begin{cases} x \sim \sigma_i = 0 \\ \bar{x} \sim \sigma_i = 1 \end{cases}. \qquad (4.2)$$

Приклад 2. Для розглянутої вище функції f_1 (див. табл. 4.1) побудуємо ДКНФ

$$f_1 = (x_1 \vee x_2 \vee x_3)(x_1 \vee \bar{x}_2 \vee \bar{x}_3)(\bar{x}_1 \vee x_2 \vee \bar{x}_3)(\bar{x}_1 \vee \bar{x}_2 \vee x_3).$$

4.3. Досконала Шефферівська нормальна форма

Наступна відома форма носить назву **досконалої Шефферівської нормальної форми** (ДШНФ). Для отримання ДШНФ в таблиці істинності відмічаються ті набори, на яких функція приймає значення 1. Конституенти одиниці записуються у вигляді термів, у яких як операція використовується штрих Шеффера. Між термами використовується також операція штрих Шеффера (4.3).

$$f(x_1, x_2, \ldots, x_n) = /_{\sigma} f(\sigma_1, \sigma_2, \ldots, \sigma_n) / x_1^{\sigma_1} / x_2^{\sigma_2} / \ldots / x_n^{\sigma_n}$$

$$\text{де} \quad \sigma_i = 0,1 \quad \text{і} \quad x_i^{\sigma_{i1}} = \begin{cases} x \sim \sigma_i = 1 \\ \bar{x} \sim \sigma_i = 0 \end{cases} \tag{4.3}$$

Приклад 3. Для функції $f_3(x_1, x_2, x_3) = (2, 3, 4)^1$ побудуємо ДШНФ:

$$f_3(x_1, x_2, x_3) = (\bar{x}_1 / x_2 / \bar{x}_3) / (\bar{x}_1 / x_2 / x_3) / (x_1 / \bar{x}_2 / \bar{x}_3).$$

4.4. Досконала Пірсівська нормальна форма

ДПНФ – **досконала Пірсівська нормальна форма**. Для отримання ДПНФ в таблиці істинності відмічаються ті набори, на яких функція приймає значення 0. Конституенти нуля записуються у вигляді термів, у яких як операція використовується стрілка Пірса. Між термами використовується також операція стрілка Пірса (4.4).

$$f(x_1, x_2, \ldots, x_n) = \downarrow_{\sigma} f(\sigma_1, \sigma_2, \ldots, \sigma_n) \downarrow x_1^{\sigma_1} \downarrow x_2^{\sigma_2} \downarrow \ldots \downarrow x_n^{\sigma_n} \downarrow$$

$$\text{де } \sigma_i = 0,1 \text{ і } x_i^{\sigma_{i1}} = \begin{cases} x \sim \sigma_i = 0; \\ \bar{x} \sim \sigma_i = 1. \end{cases} \tag{4.4}$$

Приклад 4. Для функції $f_4(x_1, x_2, x_3) = (0, 3, 7)^0$ побудуємо ДПНФ

$$f_4(x_1, x_2, x_3) = (x_1 \downarrow x_2 \downarrow x_3) \downarrow (x_1 \downarrow \bar{x}_2 \downarrow \bar{x}_3) \downarrow (\bar{x}_1 \downarrow x_2 \downarrow x_3).$$

4.5. Функціонально повні системи булевих функцій

Будь-яка булева функція може бути представлена аналітично однією з розглянутих нормальних форм. Вони використовують обмежену кількість елементарних булевих функцій. Наприклад, для ДДНФ такими функціями являються «кон'юнкція», «диз'юнкція» і «заперечення». Отже, існують системи булевих функцій, за допомогою яких можна аналітично представити будь-яку іншу, скільки завгодно складну булеву функцію. Проектування цифрових автоматів основане на використанні таких систем булевих функцій. Останнє особливо важливе для розробки комплектів інтегральних мікросхем, з яких можна побудувати довільний цифровий автомат. Проблема функціональної повноти є центральною проблемою функціональних побудов в алгебрі логіки.

Визначення. **Функціонально повною системою булевих функцій** (ФПСБФ) називається сукупність таких булевих функцій $\{f_1, f_2, ..., f_k\}$, що довільна булева функція f може бути записана у вигляді формули через функції цієї сукупності.

Виходячи з визначення ДДНФ, ДКНФ, ДШНФ і ДПНФ, до функціонально повних систем булевих функцій слід віднести системы: $\{\wedge,\vee, HI\},\{I-HI\},\{АБО-HI\}$.

Це обумовлює доцільність постановки задачі визначення властивостей, якими повинні володіти функції, що входять до складу ФПСБФ.

Рішення цієї задачі основане на понятті замкнутого відносно операції суперпозиції класу функцій. Класом булевих функцій, функціонально замкнутим відносно операції суперпозиції, є множина функцій, будь-яка суперпозиція яких дає функцію, що також належить цій множині. Серед функціонально замкнутих класів виділяють класи особливого типу, які називають передповними, і які володіють такою властивістю. Нехай передповний клас S не співпадає з множиною P можливих булевих функцій. Але якщо в нього включити будь-яку булеву функцию, що не входить в S, то новий функціонально замкнутий клас буде співпадати з множиною P. Проведені дослідження показали, що передповних класів п'ять, а для побудови ФПСБФ необхідно і достатньо, щоб її функції не містились повністю ні в одному з п'яти передповних класів.

Перелічимо передповні класи булевих функцій:

1) булеві функції, які зберігають константу 0;

2) булеві функції, які зберігають константу 1;

3) самодвоїсті булеві функції;

4) лінійні булеві функції;

5) монотонні булеві функції.

Визначення. До булевих функцій, які зберігають константу 0, належать такі булеві функції $f(x_1, ..., x_n)$, для яких справедливе співвідношення $f(0, ..., 0) = 0$.

Таблиця 4.2 – Таблиця істинності для функцій 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	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

Прикладами булевих функцій, які зберігають константу 0, являються функції f_0, f_1, …, f_7 (табл. 4.2). Оскільки таблиця істинності для функцій, які зберігають константу 0, у першому рядку значень функцій містить 0, то є рівно $2^{(2^n-1)}$ таких функцій.

Визначення. До булевих функцій, які зберігають константу 1, відносять такі булеві функції $f(x_1, ..., x_n)$, для яких справедливо: $f(1, ..., 1) = 1$.

Прикладами булевих функцій, які зберігають константу 1, являються функції f_1, f_3, f_5, f_7, f_9, f_{11}, f_{13} і f_{15} (табл. 4.2). Оскільки таблиця істинності для функцій, які зберігають константу 1, в останньому рядку значень функцій містить 1, то є рівно $2^{(2^n-1)}$ таких функцій. Перед тим, як ввести поняття класу самодвоїстих булевих функцій, дамо таке визначення.

Визначення. Булеві функції $f_1(x_1, ..., x_n)$ і $f_2(x_1, ..., x_n)$ називаються двоїстими одна відносно одної, якщо виконується співвідношення $f_1(x_1, ..., x_n) = \overline{f_2(\bar{x}_1, \bar{x}_2, ..., \bar{x}_n)}$.

Двоїстими є функції f_0 і f_{15}, f_1 і f_7, f_2 і f_{11} і т. д. (табл. 4.2).

Визначення. До самодвоїстих булевих функцій відносять такі булеві функції, які двоїсті відносно до самих себе, тобто справедливе співвідношення

$$f(x_1, ..., x_n) = \overline{f(\bar{x}_1, \bar{x}_2, ..., \bar{x}_n)}.$$

Якщо домовитись називати протилежними наборами набір $<\gamma_1, \gamma_2, ..., \gamma_n>$ і набір $<\bar{\gamma}_1, \bar{\gamma}_2, ..., \bar{\gamma}_n>$, то визначення самодвоїстих функцій буде таким.

Визначення. Булева функція називається самодвоїстою, якщо на будь-яких двох протилежних наборах вона приймає протилежні значення.

Самодвоїстими є функції f_3, f_5, f_{10}, f_{12} (табл. 4.2). Із визначення самодвоїстої функції випливає, що вона повністю визначається своїми значеннями на першій половині рядків таблиці істинності. Тому число всіх самодвоїстих булевих функцій $f(x_1, ..., x_n)$ дорівнює $2^{(2^n-1)} = \sqrt{2^{2^n}}$.

Визначення. До лінійних булевих функцій відносять такі булеві функції, які можуть бути представлені у вигляді

$$f(x_1, ..., x_n) = c_0 \oplus c_1 x_1 \oplus ... \oplus c_n x_n,$$

де $c_i \in \{0,1\}$, а $\oplus$ – операція додавання за модулем два.

Лінійними являються булеві функції $f_0, f_3, f_5, f_6, f_9, f_{10}, f_{12}, f_{15}$ (табл. 4.2), оскільки

$$f_0 = 0, f_3 = x_1, f_5 = x_2,$$

$f_6 = x_1 \oplus x_2$, $f_9 = x_1 \oplus x_2 + 1$, $f_{10} = \bar{x}_2 = 1 \oplus x_2$, $f_{12} = \bar{x}_1 = 1 \oplus x_1$, $f_{15} = 1$.

Оскільки лінійна функція однозначно залежить від задання коефіцієнтів $c_0, c_1, \ldots, c_n$, то число лінійних функцій дорівнює $2^{(n+1)}$.

Перш ніж ввести поняття класу монотонних булевих функцій, дамо таке визначення.

Визначення. Двійковий набір $\alpha =< \alpha_1, \alpha_2, \ldots, \alpha_n >$ не менше двійкового набору $\beta =< \beta_1, \beta_2, \ldots, \beta_n >$, (тобто $\alpha \geq \beta$), якщо для кожної пари $(\alpha_i, \beta_i), i \in \overline{1,n}$ справедливе співвідношення $\alpha_i \geq \beta_i$.

Так, набір 1011 > 1010. Але набори 1011 і 0100 не можна порівнювати в тому розумінні, що для них не виконуються співвідношення ні $\alpha \geq \beta$, ні $\beta \geq \alpha$.

Визначення. Булева функція f (x_1, ..., x_n) називається монотонною, якщо для будь-яких двох наборів $\alpha =< \alpha_1, \alpha_2, \ldots, \alpha_n >$ і $\beta =< \beta_1, \beta_2, \ldots, \beta_n >$, таких, що $\alpha \geq \beta$ має місце нерівність $f(\alpha_1, \alpha_2, \ldots, \alpha_n) \geq f(\beta_1, \beta_2, \ldots, \beta_n)$. Монотонними являються булеві функції f_0, f_1, f_3, f_5, f_7, f_{15} (табл. 4.4.). Разом з тим функція f_2 із табл. 4.2 не є монотонною, тому що f_2 (1,0) > f_2 (1, 1), хоча набір <1, 0> менше, ніж набір <1, 1>. Наведемо без доказу два формулювання теореми про функціональну повноту.

Теорема ПОСТА. Для того щоб система S булевих функцій була функціонально повною, необхідно і достатньо, щоб ця система містила хоча б одну булеву функцію, яка не зберігає константу 0, хоча б одну булеву функцію, яка не зберігає константу 1, хоча б одну несамодвоїсту булеву функцію, хоча б одну нелінійну булеву функцію і хоча б одну немонотонну булеву функцію.

Система булевих функцій є функціонально повною тоді і тільки тоді, коли вона цілком не міститься ні в одному з передповних класів.

Розглянемо приклади ФПСБФ. Для зручності елементарні булеві функції двох змінних і деякі булеві функції однієї змінної зведені в табл. 4.3, в якій виконана класифікація кожної з функцій за ознаками приналежності до передповних класів. Із табл. 4.3 видно, що кожна з функцій f_8 і f_{14} являється ФПСБФ. Іншими словами, використовуючи, наприклад, тільки булеву функцію f_{14} – «штрих Шеффера», можна записати у вигляді формули будь-яку булеву функцію. Табл. 4.3 дозволяє отримати й інші ФПСБФ. Ознакою функціональної повноти являється відсутність знака «+» в кожному стовпчику табл. 4.3 хоча б для однієї із булевих функцій, які складають систему. До таких ФПСБФ, найбільш поширених у практиці побудови цифрових автоматів, слід віднести: $\{\vee, \wedge, HI\}$; $\{\wedge, \oplus, HI\}$; $\{\wedge, \oplus, 1\}$; $\{\vee, HI\}$; $\{\wedge, HI\}$, де символами $\vee, \wedge, \oplus, HI, 1$ позначені булеві функції: «диз'юнкція», «кон'юнкція», «сума за модулем два», «заперечення», і «константа 1» відповідно.

Введене поняття двоїстих булевих функцій дозволяє сформулювати принцип двоїстості, який полягає у такому. Якщо формула $U = C[f_1, f_2, \ldots, f_n]$ реалізує булеву функцію $f = f(x_1, x_2, \ldots, x_n)$, то формула $U^* = C[f_1^*, f_2^*, \ldots, f_n^*]$, отримана із U заміною функцій f_1, f_2, ..., fn на двоїсті функції f_1*, f_2*, …, fn*, відповідно реалізує функцію $f^* = f^*(x_1, x_2, \ldots, x_n)$, двоїсту функції f. Формулу U

називають двоїстою U^*. Для формул над множиною $\{0,1,\bar{x}, x_1 \& x_2, x_1 \vee x_2\}$ принцип двоїстості може бути сформульований так: для отримання формули U^*, двоїстої до формули U, достатньо в формулі U всюди замінити 0 на 1, 1 на 0, і операції & на $\vee$ та $\vee$ на &.

Приклад. Із співвідношення $\overline{x_1 x_2} = \bar{x}_1 \vee \bar{x}_2$ за рахунок використання принципу двоїстості отримується співвідношення $\overline{x_1 \vee x_2} = \bar{x}_1 \bar{x}_2$. Принцип двоїстості дозволяє майже вдвічі скоротити зусилля на виведення співвідношень під час розгляду властивостей елементарних булевих функцій.

Таблиця 4.3 – Характеристики функцій двох змінних

Функція	Види функцій				
	монотонна	лінійна	самодвоїста	та, що зберігає 0	та, що зберігає 1
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}	+	+			+

Контрольні запитання

1. У чому полягає операція суперпозиції?
2. Наведіть визначення для конституенти одиниці і конституенти нуля.
3. Які особливості подання булевої функції у вигляді:
   - досконалої диз'юнктивної нормальної форми (ДДНФ);
   - досконалої кон'юнктивної нормальної форми (ДКНФ);
   - досконалої Шефферівської нормальної форми (ДШНФ);
   - досконалої Пірсівської нормальної форми (ДПНФ).
4. Побудуйте ДДНФ для такої функції $f(x_1, x_2, x_3) = (0, 1, 2, 3, 7)^1$.
5. Побудуйте ДКНФ для такої функції $f(x_1, x_2, x_3, x_4,) = (0, 1, 2, 3, 12,13,14,15)^0$.

6. Побудуйте ДШНФ для такої функції $f(x_1, x_2, x_{3,} x_{4,}) = (0, 1, 2, 3, 7,15)^1$.
7. Побудуйте ДПНФ для такої функції $f(x_1, x_2, x_3) = (0, 1, 2, 3, 7)^1$.
8. Дайте визначення для системи булевих функцій, яка є функціонально повною.
9. Перелічіть системи булевих функцій, які є функціонально повними.
10. Перелічіть передповні класи булевих функцій і дайте визначення для кожного з них.
11. У чому полягає теорема Поста?
12. Прокоментуйте зміст табл. 4.3.

5. МІНІМІЗАЦІЯ БУЛЕВИХ ФУНКЦІЙ

5.1. Карти Карно

Визначення.. Мінімальною нормальною формою (НФ) булевої функції називається НФ, яка містить найменшу кількість букв (по відношенню до всіх інших НФ, що подають задану булеву функцію).

Найбільш поширені такі мінімальні нормальні форми: диз'юнктивна, кон'ъюнктивна, Шефферівська, Пірсівська. Для отримання мінімальних форм булевих функцій використовують методи: Квайна, Квайна-Мак-Класкі, Блейка-Порецького, Нельсона, метод діаграм Вейча (або карт Карно). В основі практично всіх методів лежить закон склеювання. Найбільш поширеним є метод карт Карно.

В основі методу карт Карно лежить завдання булевих функцій діаграмами деякого спеціального виду, які отримали назву карт Карно. Карта Карно це таблиця, кількість клітинок в якій дорівнює $N = 2^n$, де n – кількість змінних, від яких залежить функція. Так, для функції $f(x_1, x_2, x_3)$, n дорівнює трьом. Отже в цьому випадку карта Карно буде мати вісім клітинок (табл. 5.1).

Таблиця 5.1 – Карта Карно для трьох змінних

x_1

x_2				
	110	111	011	010
	100	101	001	000

x_3

Суть позначень на карті (табл. 5.1) така:

• у всіх стовпцях, помічених лінією зверху (в таблиці це перший і другий стовпці), змінна x_1 має значення 1, у всіх інших – 0.

• у всіх рядках, помічених лінією (перший рядок в таблиці), змінна x_2 має значення 1, у всіх інших – 0.

• у всіх стовпцях, помічених лінією знизу (в таблиці це другий і третій стовпці), змінна x_3 має значення 1, у всіх інших – 0.

В кожній клітинці таблиці значення змінних x_1, x_2 і x_3 відповідають положенню клітинки в таблиці. В кожній клітинці таблиці (табл. 5.1) наведені набори двійкових значень змінних x_1, x_2 і x_5.

Карти Карно мають таку властивість: кожна сусідня клітинка відрізняється значенням тільки по одній змінній. Сусідні клітинки це клітинки, розміщені по відношенню до даної зверху, знизу, зліва і праворуч з

урахуванням граничних клітинок. Так, клітинки 110 і 111 (табл. 5.1) відрізняються значенням тільки змінної x_5.

Якщо набір двійкових значень змінних x_1, x_2 і x_3 в кожній клітинці таблиці перевести в десятковий код, то отримаємо десяткові порядкові номери наборів функції. В табл. 5.2 наведені порядкові номери наборів функції та занесена функція $f(x_1, x_2, x_3) = (1,6,7)$[1].

Таблиця 5.2 – Карта Карно з десятковими номерами клітинок і одиничними значеннями функції f

x_1

x_2				
	1 6	1 7	3	2
	4	5	1 1	0

x_3

В клітинках з номерами 1, 6, 7 проставляємо одиниці. Нульові значення булевої функції в карті Карно можна не відображати.

Для карт Карно характерне таке:

- кожній клітинці карти відповідає свій двійковий набір;
- сусідні двійкові набори розміщені поруч в рядку або в стовпчику;
- стовпці, а також рядки, розміщені по краях карти, також вважаються сусідніми.

Для функції $f(x_1, x_2, x_3, x_4)$ чотирьох змінних карта Карно наведена в табл.5.3 і в табл.5.4.

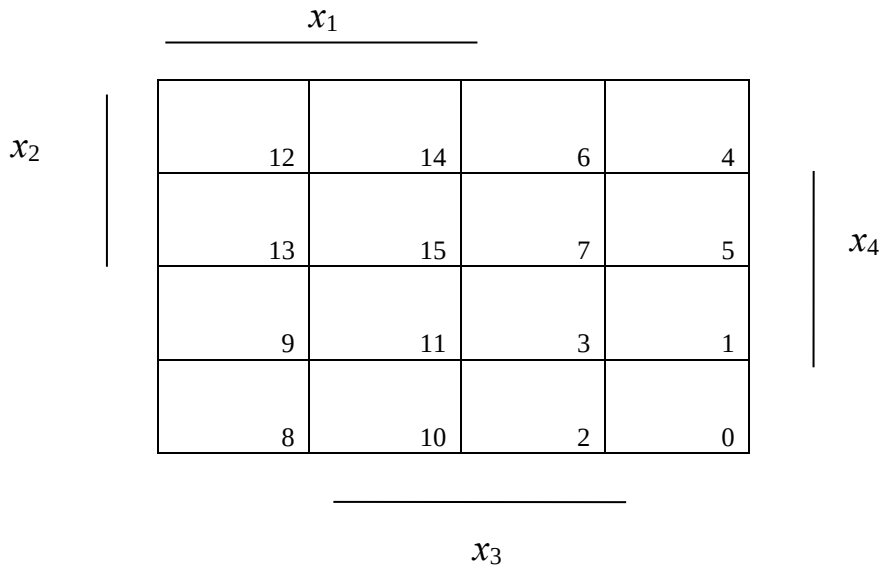
Таблиця 5.3 – Карта Карно для чотирьох змінних

x_1

x_2				x_4
	1100	1110	0110	0100
	1101	1111	0111	0101
	1001	1011	0011	0001
	1000	1010	0010	0000

x_3

Таблиця 5.4 – Карта Карно для чотирьох змінних з номерами наборів



Карти Карно для функцій з кількістю змінних більше чотирьох використовуються рідко.

Так як сусідні набори відрізняються тільки однією компонентою, то конституенти, які відповідають таким наборам, склеюються. Наприклад, для функції, заданої табл. 5.2, конституенти, що відповідають одиницям в клітинках 6 і 7 таблиці, склеюються і породжують елементарний добуток із двох букв:

$$x_1 x_2 \bar{x}_3 \vee x_1 x_2 x_3 = x_1 x_2$$

Відмітимо, що отримуваний елементарний добуток легко визначити зразу за картою: це добуток змінних, які приймають одне і те ж значення в обох клітинках.

Карти Карно являються зручним візуальним засобом мінімізації булевих функцій. У них сусідні клітинки об'єднуються в **правильні конфігурації** (ПК). У правильні конфігурації можна об'єднувати 2^i сусідніх клітинок, тобто 1, 2, 4, 8 або всі 16 (для карт із чотирма змінними). Раніш ніж перейдемо до опису процедури отримання ПК, введемо поняття рангу правильної конфігурації.

Рангом правильної конфігурації є число $n = log_2 N$, де N – кількість клітинок, об'єднаних у правильну конфігурацію.

Загальне правило склеювання на картах Карно можна сформулювати таким чином:

• склеюванню підлягають прямокутні конфігурації, заповнені одиницями або нулями, ті, що містять кількість клітинок, яка відповідає степеню числа два.

• під час об'єднання клітинок прагнуть отримати мінімально можливу кількість правильних конфігурацій максимального рангу. Ранг правильної конфігурації показує за скількома змінними відбувається склеювання.

Кількість m змінних функції, які залишились в елементарному добутку після склеювання,

$$m = M - n,$$

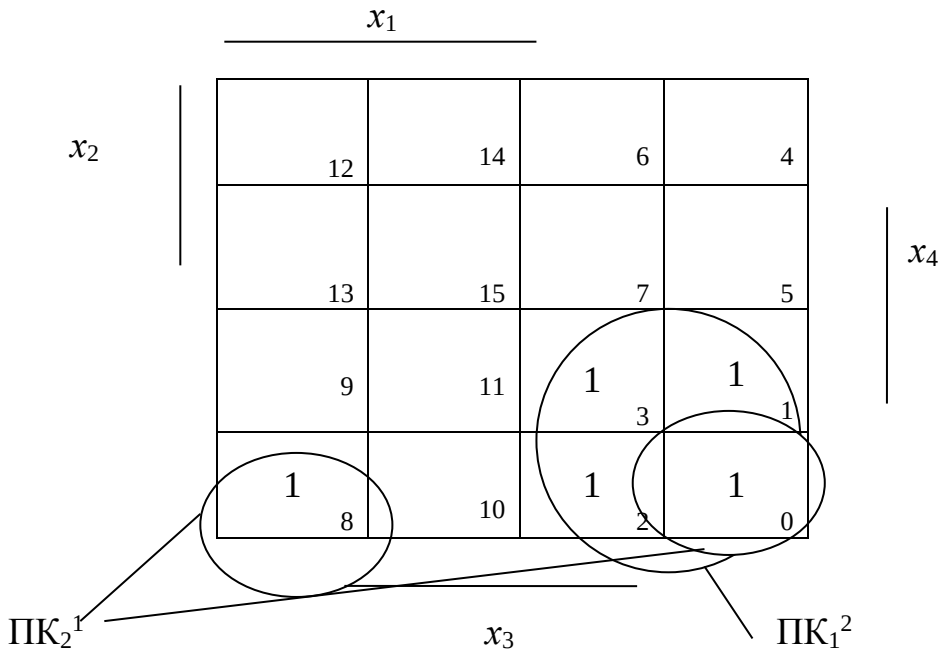
де M – кількість змінних функції; n – кількість наборів, що склеюються.

Правильні конфігурації мають індекси – нижній і верхній. Нижній індекс означає порядковий номер правильної конфігурації, верхній – її ранг.

Правильні конфігурації описуються таблицею ПК. Стовпці таблиці мають назви змінних, від яких залежить функція f, а рядки – назви всіх отриманих ПК. На перетині рядка для вибраної в цій таблиці ПК з кожним із стовпчиків x_i (i – порядковий номер змінної x) записується те значення, яке приймає змінна x_i (0 або 1). Якщо якась змінна для даної ПК при цьому приймає різні значення, то в таблиці у відповідній клітинці ставиться дефіс. Говорять, що по цій змінній відбувається склеювання.

Цей метод широко використовується на практиці завдяки простоті і зручності. Мінімізація булевої функції полягає в знаходженні мінімальної кількості ПК, які повинні покривати всі одиниці (нулі) блоками із одиниць (нулів) максимальної конфігурації, розміщених в сусідніх клітинках карти. При цьому завжди вважається, що лівий край карти Карно примикає до її правого краю, а верхній – примикає до її нижнього краю.

Приклад 1. Задана функція $f_1(x_1, x_2, x_3, x_4) = (0, 1, 2, 3, 8)^1$. Занести функцію в карту Карно і побудувати правильні конфігурації за одиничними значеннями функції.



Об’єднуємо в $ПК_1$ клітинки 0, 1, 2 і 3. Так як об’єднані чотири клітинки, то ранг $ПК_1$ буде дорівнювати двом. Об’єднуємо в $ПК_2^1$ клітинки 0 і 8. Ранг $ПК_2$ дорівнює одиниці.

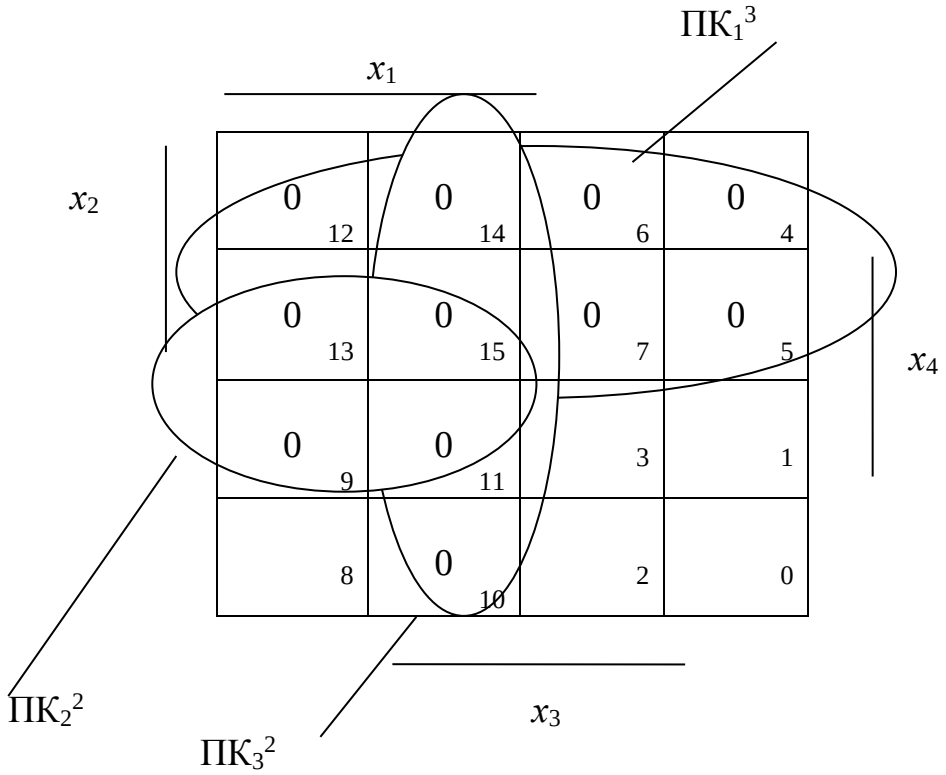
Описуємо отримані ПК за допомогою табл. 5.5.

Таблиця 5.5 – Опис значень змінних для правильних конфігурацій за одиничними значеннями функції f_1

Правильні конфігурації	Значення змінних			
	x_1	x_2	x_3	x_4
$ПК_1^2$	0	0	–	–
$ПК_2^1$	–	0	0	0

Для всіх клітинок карти Карно, які входять в $ПК_1^2$, змінні x_1 і x_2 приймають нульові значення. У відповідних стовпцях табл. 5.5 записуємо нулі. Змінна x_3 в клітинках 2 і 3 приймає одиничне значення, а в клітинках 0 і 1 – нульове значення. Тому в стовпець x_3 для $ПК_1^2$ вписуємо дефіс. Змінна x_4 також приймає різні значення в указаних клітинках, тому в стовпець x_4 також вписуємо дефіс.

Приклад 2. Для функції $f_1(x_1, x_2, x_3, x_4) = (0, 1, 2, 3, 8)^1$ побудувати правильні конфігурації за нульовими значеннями функції f_1.



Об’єднаємо в $ПК_1$ клітинки 4, 5, 6, 7, 12, 13, 14 і 15. Так як об’єднані вісім клітинок, то ранг буде рівнятись трьом, тобто маємо $ПК_1^{3.}$ Об’єднаємо в $ПК_2$ клітинки 9, 11, 13 і 15. Ранг дорівнює двом і маємо $ПК_2^2$. Об’єднаємо в $ПК_3$ клітинки 10, 11, 14 і 15. Отримаємо $ПК_3^2$.

Описуємо значення змінних для правильних конфігурацій за нульовими значенням функції f_1 за допомогою табл. 5.6.

Таблиця 5.6 – Опис значень змінних для правильних конфігурацій за нульовими значеннями функції f_1

Правильні конфігурації	Значення змінних			
	x_1	x_2	x_3	x_4
$ПК_1^3$	–	1	–	–
$ПК_2^2$	1	–	–	1
$ПК_3^2$	1	–	1	–

5.2. Мінімальна диз'юнктивна нормальна форма

Мінімальна диз'юнктивна нормальна форма (МДНФ) записується як диз'юнкція елементарних кон'юнкцій, що відповідають виділеним блокам одиниць на карті Карно.

Для отримання МДНФ необхідно виконати такі дії:

• заносимо одиничні значення функції в карту Карно;

• створюємо ПК за **одиничними** значеннями. Одиниці, які ввійшли на попередніх кроках в одну із кон'юнкцій, вважаємо **покритими**. Із непокритих і покритих одиниць створюємо наступну кон'юнкцію, покриваючи максимальну кількість непокритих одиниць. Нову кон'юнкцію тільки із покритих одиниць створювати не потрібно. Процес завершується, коли покриті всі одиниці;

• описуємо всі ПК за допомогою таблиці;

• для кожної ПК складаємо кон'юнкцію змінних, від яких вона залежить.

Якщо змінна для даної ПК приймає одиничне значення, то вона береться в прямому вигляді, а інакше – в інверсному:

$$\wedge x_i^{\sigma_i} = \begin{cases} x, \alpha_i = 1 \\ \bar{x}, \alpha_i = 0 \end{cases}$$

• об'єднуємо за допомогою операції диз'юнкції всі описані ПК:

$$f(x_1, x_2, ..., x_n) = \bigvee_{\alpha} f(\alpha_1, \alpha_2, ..., \alpha_n) \cdot x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdot ... \cdot x_n^{\alpha_n}. \qquad (5.1)$$

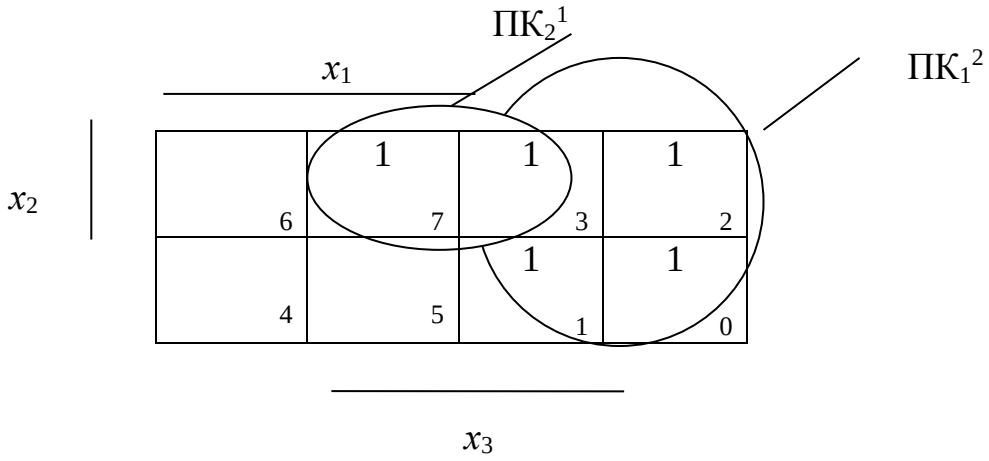
й отримуємо мінімальну диз'юнктивну нормальну форму.

Приклад 3. Функція f_3 задана таблицею істинності (табл. 5.7). Побудувати МДНФ.

Таблиця 5.7 – Таблиця істинності функції f_3

x_1	x_2	x_3	f_3
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Заносимо одиничні значення функції f_3 в карту Карно.



Об’єднуємо клітинки 0, 1, 2 і 3 в $ПК_1{}^2$, а клітинки 3 і 7 в $ПК_2{}^1$. Правильні конфігурації описуємо за допомогою табл. 5.8.

Таблиця 5.8 – Опис значень змінних для правильних конфігурацій за одиничними значеннями функції f_3

Правильні Конфігурації	Значення змінних		
	x_1	x_2	x_3
$ПК_1{}^2$	0	–	–
$ПК_2{}^1$	–	1	1

$ПК_1{}^2$ описується тільки однією змінною, а саме $\overline{x_1}$. $ПК_2{}^1$ описується добутком змінних x_2 і x_3 Об’єднавши за допомогою операції диз’юнкції отримані описи ПК, одержимо МДНФ:

$$f_{3\text{мднф}} = \overline{x_1} \vee x_2 \cdot x_3.$$

5.3. Мінімальна кон'юнктивна нормальна форма

Мінімальна кон'юнктивна нормальна форма (МКНФ) записується як кон'юнкція елементарних диз'юнкцій , що відповідають виділеним блокам нулів на карті Карно.

Для отримання МКНФ необхідно виконати такі дії:

• заносимо нульові значення функції в карту Карно;

• створюємо ПК за **нульовими** значеннями. Нулі, які ввійшли на попередніх кроках в одну із диз'юнкцій, вважаємо **покритими**. Із непокритих і покритих нулів створюємо наступну диз'юнкцію, покриваючи максимальну кількість непокритих нулів. Нову диз'юнкцію тільки із покритих нулів створювати не потрібно. Процес завершується тоді, коли покриті всі нулі;

• описуємо всі ПК за допомогою таблиці;

• для кожної ПК складаємо диз'юнкцію змінних, від яких вона залежить.

Якщо змінна для даної ПК приймає нульове значення, то вона береться в прямому вигляді, а інакше – в інверсному:

$$\vee\, x_i^{\alpha_i} = \begin{cases} x, \alpha_i = 0 \\ \overline{x}, \alpha_i = 1 \end{cases}$$

• об'єднуємо за допомогою операції кон'юнкції всі описані ПК:

$$f(x_1, x_2, \ldots, x_n) = \bigwedge_{\alpha} f(\alpha_1, \alpha_2, \ldots, \alpha_n) \vee x_1^{\alpha_1} \vee x_2^{\alpha_2} \vee \ldots \vee x_n^{\alpha_n} \qquad (5.2)$$

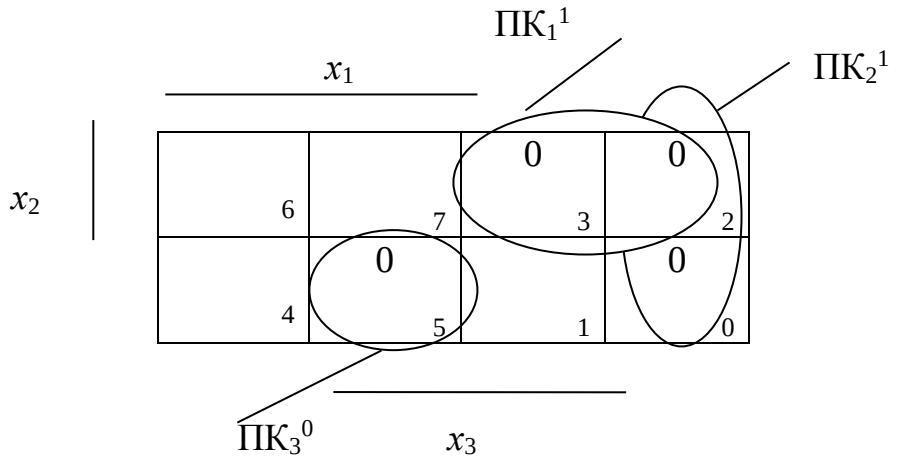
й отримуємо мінімальну кон'юнктивну нормальну форму.

Приклад 4. Функція f_4 задана таблицею істинності (табл.5.9). Побудувати МКНФ.

Таблиця 5.9 – Таблиця істинності функції f_4

x_1	x_2	x_3	f_4
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Заносимо нульові значення функції f_4 в карту Карно.



Об’єднуємо клітинки 2 і 3 в $ПК_1^1$, клітинки 0 і 2 – в $ПК_2^1$, а клітинку 5 – в $ПК_3^0$. Правильні конфігурації описуємо за допомогою табл. 5.10.

Таблиця 5.10 – Опис значень змінних для правильних конфігурацій за нульовими значеннями функції f_4

Правильні конфігурації	Значення змінних		
	x_1	x_2	x_3
$ПК_1^1$	0	1	–
$ПК_2^1$	0	–	0
$ПК_3^0$	1	0	1

$ПК_1^1$ описується як диз’юнкція змінних $x_1 \vee \overline{x_2}$. $ПК_2^1$ описується як диз’юнкція змінних $x_1 \vee x_3$. $ПК_3^0$ описується як диз’юнкція змінних $\overline{x_1} \vee x_2 \vee \overline{x_3}$. Об’єднавши за допомогою операції кон’юнкції одержані описи ПК, отримаємо МКНФ:

$$f_{4мкнф} = (x_1 \vee \overline{x_2})\,(x_1 \vee x_3)\,(\overline{x_1} \vee x_2 \vee \overline{x_3}).$$

5.4. Мінімальна Шефферівська нормальна форма

Операція штрих Шеффера або операція ***І-НІ*** являється функціонально-повним базисом. Тому за допомогою операції штрих Шеффера може бути подана будь-яка булева функція.

Для отримання мінімальної Шефферівської нормальної форми (МШНФ) необхідно виконати такі дії:

• заносимо одиничні значення функції в карту Карно;

• створюємо ПК за **одиничними** значеннями. Одиниці, які ввійшли на попередніх кроках в одну із кон’юнкцій, вважаємо **покритими**. Із непокритих і

покритих одиниць створюємо наступну ПК, покриваючи максимальну кількість непокритих одиниць. Процес завершується, коли покриті всі одиниці;

• описуємо всі ПК за допомогою таблиці;

• змінні, від яких залежить ПК об'єднуємо за допомогою операції штрих Шеффера. Якщо змінна для даної ПК приймає одиничне значення, то вона береться в прямому вигляді, а інакше – в інверсному:

$$/\ x_i^{\sigma_i} = \begin{cases} x, \alpha_i = 1 \\ \overline{x}, \alpha_i = 0 \end{cases}$$

Виняткові ситуації:

✓ якщо ПК має максимальний ранг (тобто описується тільки однією змінною), то змінна, яка описує ПК, береться в інверсному вигляді;

✓ якщо має місце тільки одна ПК, то береться загальна інверсія над усіма змінними, які описують цю ПК.

Об'єднуємо за допомогою операції штрих Шеффера всі описані ПК:

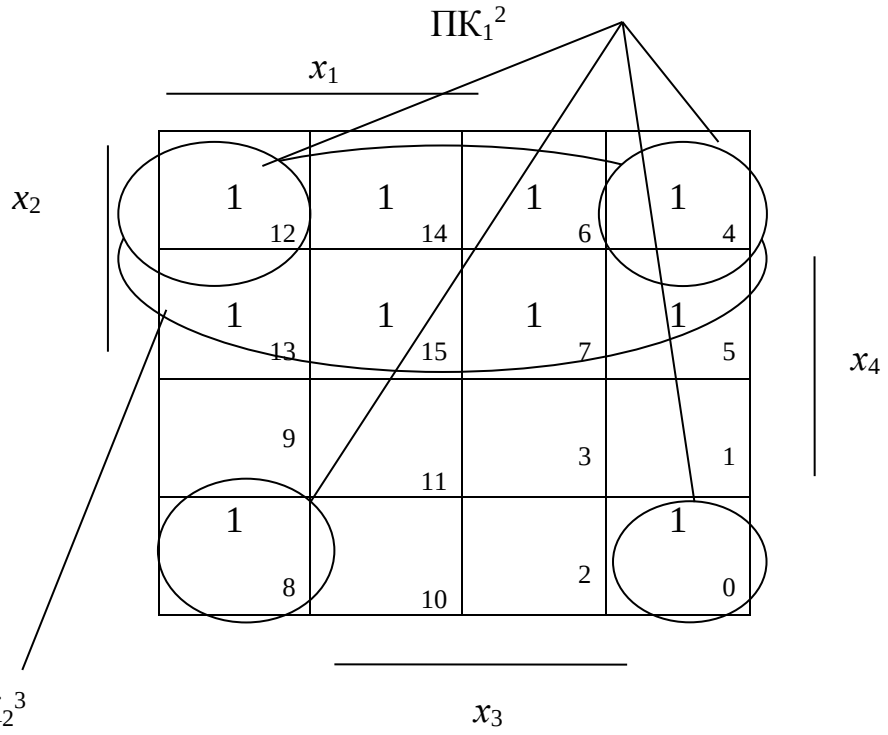
$$f(x_1, x_2, \ldots, x_n) = /\underset{\alpha}{} f(\alpha_1, \alpha_2, \ldots, \alpha_n) / x_1^{\alpha_1} / x_2^{\alpha_2} / \ldots / x_n^{\alpha_n}$$

і отримуємо мінімальну Шефферівську нормальну форму.

Приклад 5. Задана функція $f_5(x_1, x_2, x_3, x_4)$ = (0, 4, 5, 6, 7, 8, 12, 13, 14, 15)[1]. Побудувати МШНФ.

Заносимо одиничні значення функції f_5 в карту Карно.

Об'єднуємо клітинки 0, 4 , 8 і 12 в $ПК_1^2$, а клітинки 4, 5, 6, 7, 12, 13, 14 і 15 в $ПК_2^3$



Правильні конфігурації описуємо за допомогою табл. 5.11.

Таблиця 5.11 – Опис значень змінних для правильних конфігурацій за одиничними значеннями функції f_5

Правильні конфігурації	Значення змінних			
	x_1	x_2	x_3	x_4
$ПК_1^2$	–	–	0	0
$ПК_2^3$	–	1	–	–

$ПК_1^2$ описується як – $\bar{x}_3 / \bar{x}_4$. Для $ПК_2^3$ має місце виняткова ситуація. Тому $ПК_2^3$ описується як $\overline{x_2}$. Об'єднавши за допомогою операції штрих Шеффера отримані описи ПК, отримаємо МШНФ:

$$f_{5\text{мшнф}} = (\bar{x}_3 / \bar{x}_4) / \overline{x_2}.$$

5.5. Мінімальна Пірсівська нормальна форма

Операція стрілка Пірса або операція ***АБО-НІ*** являється функціонально-повним базисом. Тому за допомогою операції стрілка Пірса може бути подана будь-яка булева функція.

Для отримання мінімальної Пірсівської нормальної форми (МПНФ) необхідно виконати такі дії:

• заносимо нульові значення функції в карту Карно;

• створюємо ПК за **нульовими** значеннями. Нулі, які ввійшли на попередніх кроках в одну із ПК, вважаємо **покритими**. Із непокритих і покритих нулів створюємо наступну ПК, стараючись при цьому покрити максимальну кількість непокритих нулів. Нову ПК тільки із покритих нулів створювати не треба. Процес завершується, коли покриті всі нулі;

• описуємо всі ПК за допомогою таблиці;

• змінні, від яких залежить ПК, об'єднуємо за допомогою операції стрілка Пірса. Якщо змінна для даної ПК приймає одиничне значення, то вона береться в інверсному вигляді, інакше – в прямому:

$$\downarrow x_i^{\sigma_i} = \begin{cases} x, \alpha_i = 0 \\ \bar{x}, \alpha_i = 1 \end{cases}$$

Виняткові ситуації:

✓ якщо ПК має максимальний ранг (тобто описується тільки однією змінною), то змінна, яка описує ПК, береться в інверсному вигляді;

✓ якщо має місце тільки одна ПК, то береться загальна інверсія над усіма змінними, які описують цю ПК.

Об'єднуємо за допомогою операції стрілка Пірса всі описані ПК:

$$f(x_1, x_2, \ldots, x_n) = \underset{\sigma}{\downarrow} f(\alpha_1, \alpha_2, \ldots, \alpha_n) \downarrow x_1^{\alpha_1} \downarrow x_2^{\alpha_2} \downarrow \ldots \downarrow x_n^{\alpha_n}$$

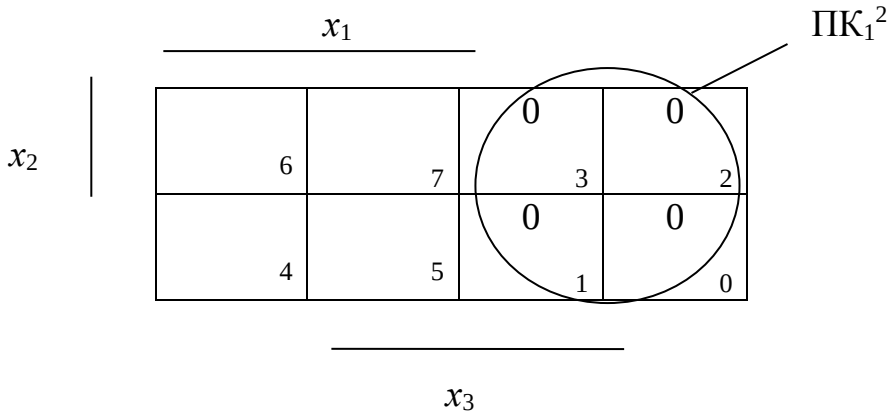
й отримуємо мінімальну Пірсівську нормальну форму.

Приклад 6. Функція f_6 задана таблицею істинності (табл. 5.12). Побудувати МПНФ.

Таблиця 5.12 – Таблиця істинності функції f_6

x_1	x_2	x_3	f_6
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Заносимо нульові значення функції f_6 в карту Карно.



Об’єднуємо клітинки 0, 1, 2 і 3 в $ПК_1^2$. Правильна конфігурація $ПК_1^2$ описується за допомогою табл. 5.15.

Таблиця 5.13 – Опис значень змінних для правильної конфігурації за нульовими значеннями функції f_6

Правильна Конфігурація	Значення змінних		
	x_1	x_2	x_3
$ПК_1^2$	0	–	–

В даному випадку мають місце одночасно дві виняткові ситуації. Тому

$$f_{6\ мпнф} = \overline{\overline{x_1}} = x_1.$$

Контрольні запитання

1. Які мінімальні нормальні форми булевих функцій ви можете назвати?
2. Які особливості використання карт Карно для отримання мінімальних нормальних форм булевих функцій?
3. Що являють собою правильні конфігурації (ПК)?
4. Як отримуються порядкові набори клітин карти Карно?
5. Які клітини карти Карно є сусідніми?
6. Що таке ранг правильної конфігурації?
7. Як обчислюється кількість змінних які описують ПК.
8. Як описуються правильні конфігурації?
9. Які дії треба виконати, щоб отримати такі мінімальні нормальні форми булевих функцій: диз'юнктивну, кон'юнктивну, Шефферівську і Пірсівську?
10. Побудувати МДНФ, МКНФ, МПНФ, МШНФ для наступної функції:
    $f_5(x_1, x_2, x_3, x_4) = (1, 2, 3, 4, 7, 8, 12, 13, 14, 15)$[1].

6. АБСТРАКТНІ ЦИФРОВІ АВТОМАТИ

6.1. Основні поняття, пов'язані з абстрактними автоматами

Цифровий (дискретний) автомат – пристрій, який здійснює прийом, зберігання та перетворення дискретної інформації за деяким алгоритмом. В якомусь розумінні до автоматів можна віднести як реальні пристрої (обчислювальні машини, живі організми і т. ін.), так і абстрактні системи (математичні машини, аксіоматичні теорії та ін.).

Загальну теорію автоматів підрозділяють на *абстрактну і структурну*. Відмінність між ними полягає в тому, що абстрактна теорія, відсторонюючись від структури автомата (тобто не цікавлячись способом його побудови), вивчає лише поведінку автомата відносно зовнішнього середовища. Абстрактна теорія автоматів близька, таким чином, до теорії алгоритмів, будучи, по суті, її подальшою деталізацією. В протилежність абстрактній теорії структурна теорія цікавиться як структурою самого автомата, так і структурою вхідних дій і реакцією автомата на них. У структурній теорії вивчаються способи побудови автоматів, способи копіювання вхідних дій і реакцій автомата. Таким чином, структурна теорія автоматів являється продовженням і подальшим розвитком абстрактної теорії. Спираючись на апарат булевих функцій і на абстрактну теорію автоматів, структурна теорія дає ефективні рекомендації щодо розробки реальних пристроїв обчислювальної техніки.

Абстрактний кінцевий автомат описується трьома скінченними множинами і двома функціями: $A = \{X, Y, S, \delta, \lambda\}$,

де X – множина вхідних сигналів або вхідний алфавіт,

Y – множина вихідних сигналів або вихідний алфавіт,

S – множина станів або алфавіт станів,

δ – функція переходів, $s(t + 1) = \delta(s(t), x(t))$,

λ – функція виходів, $y(t) = \lambda(s(t), x(t))$.

Функція переходів δ автомата A задає відображення $(X \times S) \to S$. Функція виходів λ автомата A задає або відображення $(X \times S) \to Y$, або відображення $S \to Y$.

Функція переходів δ показує, що автомат A, знаходячись у деякому стані $s_i \in S$, під час появи вхідного сигналу $x_j \in X$ переходить в деякий стан $s_p \in S$. Це записується виразом $s_p = \delta(s_i, x_j)$. Функція виходів λ, яка задає відображення $(X \times S) \to Y$, показує, що автомат A, знаходячись в деякому стані $s_i \in S$, під час появи вхідного сигналу $x_j \in X$, виробляє вихідний сигнал $y_k \in Y$. Це записується виразом $y_k = \lambda(s_i, x_j)$.

Абстрактний цифровий автомат називається *ініціальним,* якщо на множині його станів S виділяється спеціальний початковий стан $s_0 \in S$, тобто ініціальний абстрактний автомат описується сукупністю шести об'єктів $\{X, Y, S, \delta, \lambda, s_0\}$. Виділення на множині S початкового стану s_0 пояснюється чисто практичними міркуваннями, пов'язаними з виникаючою часто необхідністю фіксувати умови початку роботи автомата.

За способом формування функції виходів виділяють два типи абстрактних автоматів, а саме автомат **Мілі** та автомат **Мура**. В автоматі Мілі функція виходів λ задає відображення $(X \times S) \rightarrow Y$. В автоматі Мура функція виходів λ задає відображення $S \rightarrow Y$. Довільний абстрактний автомат Мілі або Мура має один вхідний і один вихідний канали.

Виділяють повністю визначені та частково визначені автомати. **Повністю визначеним** називається абстрактний цифровий автомат, у якого функції переходів і виходів визначені для всіх пар (x_j, s_k).

Частково визначеним називається абстрактний цифровий автомат, у якого функція переходів, або функція виходів, або обидві ці функції визначені не для всіх пар (x_j, s_k).

Вважається, що абстрактний автомат функціонує в дискретному автоматному часі $t = 0, 1, 2, ...$, а переходи із стану в стан відбуваються миттєво. В кожний момент t дискретного часу автомат знаходиться в деякому стані $s(t)$ із множини станів S.

Якщо автомат ініціальний, то в початковий момент часу t, який дорівнює нулю, він завжди знаходиться в початковому стані $s(t) = s(0) = s_o$. В момент часу t, знаходячись в стані $s(t)$, автомат здатний сприйняти на вході букву вхідного алфавіту $x(t) \in X$. У відповідності до функції виходів λ він видасть у той же момент часу t букву вихідного алфавіту $y(t)$, а у відповідності до функції переходів δ перейде в наступний стан $s(t + 1)$. Згідно з визначенням абстрактного автомата, автомат Мілі в цьому випадку характеризується системою рівнянь (6.1):

$$\begin{aligned} y(t) &= \lambda[s(t), x(t)], \\ s(t + 1) &= \delta\,[s(t), x(t)], \end{aligned} \qquad (6.1)$$

а автомат Мура – системою рівнянь (6.2):

$$\begin{aligned} y(t) &= \lambda[s(t)], \\ s(t + 1) &= \delta\,[s(t), x(t)]. \end{aligned} \qquad (6.2)$$

Якщо на вхід ініціального абстрактного автомата Мілі або Мура, встановленого в початковий стан s_o, подавати буква за буквою деяку послідовність букв вхідного алфавіту $x_0, x_1, ..., x_n$ – вхідне слово, то на виході автомата будуть послідовно з'являтись букви вихідного алфавіту $y_0, y_1, ..., y_n$ – вихідне слово. Таким чином, на рівні абстрактної теорії функціонування цифрового автомата є перетворенням вхідних слів у вихідні слова.

Поняття стану у визначенні абстрактного автомата введене у зв'язку з тим, що більшість реальних процесів, якими управляють реально побудовані цифрові автомати, вимагають для свого правильного функціонування попереднього розвитку процесу в часі. Вихідний сигнал, який видає автомат у даний момент часу, визначається не тільки вхідною дією на автомат, але і станом, в якому автомат в цей момент часу знаходився. Наприклад, під час

побудови автомата з оплати послуг мобільного зв‘язку треба мати інформацію як про номер абонента, так і про суму внесеної оплати. Ця інформація забезпечується наявністю різних станів у абстрактного автомата. Абстрактні цифрові автомати, які відповідають уведеному визначенню абстрактного автомата, називають також абстрактними автоматами з пам’яттю, оскільки існування в автоматі множини різних його станів можливе тільки за наявності пам’яті в автомата.

Ряд процесів, якими управляють реальні автомати, не потребують для свого правильного функціонування знання попереднього розвитку процесу в часі. В автоматах, які управляють такими процесами, вихідний сигнал визначається тільки вхідною дією на автомат. Такі автомати називаються абстрактними автоматами з тривіальною пам’яттю або комбінаційними автоматами. Найпростішим комбінаційним автоматом можна вважати схемну реалізацію будь-якої булевої функції.

6.2. Способи задання абстрактних автоматів

Алфавіти входів, станів і виходів автоматів задаються як звичайні множини, наприклад, переліченням їх елементів. Функції переходів і виходів можуть бути задані у вигляді матриці, графічно та аналітично. Тому будь-який абстрактний автомат може бути заданий трьома способами: ***у вигляді матриці, графічно та аналітично.***

Під час використання **матричного** способу автомат визначається або двома таблицями – таблицею переходів і таблицею виходів, або повною матрицею. Часто таблиці переходів і виходів об’єднуються в одну таблицю, яка називається основною таблицею абстрактного автомата. Таблиця переходів задає відображення $(X \times S) \rightarrow S$, тобто визначає функцію переходів автомата. Таблиця виходів, залежно від типу автомата, який розглядається, задає відображення $(X \times S) \rightarrow Y$, або $S \rightarrow Y$, тобто визначає функцію виходів автомата.

Таблиця переходів довільного повністю визначеного абстрактного автомата будується таким чином (табл. 6.1). Стовпці таблиці позначаються буквами вхідного алфавіту автомата x_{jt}, а рядки таблиці – буквами алфавіту станів автомата $s_{i(t\text{-}1)}$. У клітинці таблиці переходів, що знаходиться на перетині стовпця, позначеного вхідним сигналом x_{jt}, і рядка, позначеного станом $s_{i(t\text{-}1)}$, вказується стан, який є результатом переходу автомата із стану $s_{i(t\text{-}1)}$ під впливом вхідного сигналу x_{jt}, що визначається виразом $s_{kt} = \delta\,(s_{i(t\text{-}1)}, x_{jt})$.

Таблиця 6.1 – Таблиця переходів автоматів Мілі та Мура

Стани	Входи	
	x_{1t}	x_{2t}
$s_{0(t\text{-}1)}$	s_{0t}	s_{1t}
$s_{1(t\text{-}1)}$	s_{1t}	s_{2t}
$s_{2(t\text{-}1)}$	s_{2t}	s_{0t}

Таблиці виходів автоматів Мілі та Мура розрізняються. Таблиця виходів повністю визначеного автомата Мілі будується таким чином (табл. 6.2). Ідентифікація стовпчиків і рядків, а також формат таблиці відповідають таблиці переходів повністю визначеного автомата. Але в клітинці таблиці виходів, що знаходиться на перетині стовпця, позначеного вхідним сигналом x_{jt}, і рядка, позначеного станом $s_{i(t-1)}$, вказується вихідний сигнал y_{kt}, який автомат Мілі формує, знаходячись у стані $s_{i(t-1)}$ за наявності вхідного сигналу x_{jt}, що визначається виразом $y_{kt} = \lambda\,(s_{i(t-1)}, x_{jt})$.

Таблиця 6.2 – Таблиця виходів автомата Мілі

Стани	Входи	
	x_{1t}	x_{2t}
$s_{0(t-1)}$	y_{0t}	y_{0t}
$s_{1(t-1)}$	y_{0t}	y_{0t}
$s_{2(t-1)}$	y_{0t}	y_{1t}

Таблиця виходів повністю визначеного автомата Мура (табл.6.3) будується так: кожному стану автомата приписується свій вихідний сигнал.

Таблиця 6.3 – Таблиця виходів автомата Мура

Стани	Виходи
$s_{0(t-1)}$	y_{0t}
$s_{1(t-1)}$	y_{0t}
$s_{2(t-1)}$	y_{1t}

Якщо автомат частково визначений, то в деяких клітинках його таблиці виходів може стояти дефіс, яким позначається відсутність вихідного сигналу. При цьому дефіс обов'язково ставиться в тих клітинках таблиці виходів, які відповідають таким же клітинкам з дефісом в таблиці переходів автомата Мілі. Останнє пов'язане з тим, що, якщо в частково визначеному автоматі Мілі є пара $s_{i(t-1)} \in S$ та $x_{jt} \in X$ і така, що перехід із стану $s_{i(t-1)}$ під дією вхідного сигналу x_{jt} невизначений, то невизначеним є і значення вихідного сигналу на такому (неіснуючому) переході.

Таблиці переходів і виходів автомата часто поєднуються в одну таблицю, яка називається **основною таблицею** автомата. Якщо об'єднати в одну таблицю табл. 6.1 і табл. 6.2, то отримаємо основну таблицю автомата Мілі.

Таблиця 6.4 – Основна таблиця автомата Мілі

Стани	Входи	
	x_{1t}	x_{2t}
$s_{0(t-1)}$	s_{0t}/y_{0t}	s_{1t}/y_{0t}
$s_{1(t-1)}$	s_{1t}/y_{0t}	s_{2t}/y_{0t}
$s_{2(t-1)}$	s_{2t}/y_{0t}	s_{0t}/y_{1t}

Окрім розглянутих вище таблиць переходів і виходів, довільний абстрактний автомат може бути описаний **повною матрицею** або матрицею з'єднань (табл. 6.5). Такий опис – один із способів матричного задання абстрактних автоматів. Повна матриця довільного абстрактного автомата є квадратною і має стільки стовпців (рядків), скільки різних станів має автомат, що розглядається. Кожний стовпчик (рядок) матриці з'єднань позначається буквою стану автомата. В клітинці повної матриці, яка знаходиться на перетині стовпця, позначеного буквою стану s_{jt}, і рядка, позначеного буквою стану $s_{i(t-1)}$ автомата, записується буква вхідного сигналу x_{pt} (або диз'юнкція вхідних сигналів), під дією якого (або яких) відбувається перехід автомата із стану $s_{i(t-1)}$ в стан s_{jt}. Якщо матрицею з'єднань задається абстрактний автомат Мілі, то поряд з буквою вхідного сигналу x_{pt} через косу лінію вказується буква вихідного сигналу y_{kt} який автомат Мілі формує, здійснюючи перехід $s_{jt} = \delta\,(s_{i(t-1)}, x_{jt})$. Якщо матрицею з'єднань задається абстрактний автомат Мура, то вихідними сигналами позначаються стани автомата, які ідентифікують рядки повної матриці.

Таблиця 6.5 – Повна матриця автомата Мілі

Стани автомата $s_{i(t-1)}$	Стани автомата s_{jt}		
	s_{0t}	s_{1t}	s_{2t}
$s_{0(t-1)}$	x_{1t}/y_{0t}	x_{1t}/y_{1t}	–
$s_{1(t-1)}$	–	x_{1t}/y_{0t}	x_{2t}/y_{0t}
$s_{2(t-1)}$	x_{2t}/y_{1t}	–	x_{1t}/y_{0t}

При графічному способі задання абстрактні автомати подаються орієнтованими графами: стани автомата відображаються вершинами графа, а переходи між станами – дугами між відповідними вершинами. При цьому кожній дузі графа приписується деяка буква x_i вхідного алфавіту автомата, яка вказує, що даний перехід автомата відбувається тільки при появі вхідного сигналу x_i, а кожній вершині графа – буква відповідного стану автомата. Якщо графом відображається автомат Мілі, то вихідні сигнали автомата проставляються на дугах графа (у відповідності до таблиці виходів автомата) поряд з буквою вхідного сигналу. Якщо графом відображається автомат Мура, то вихідні сигнали автомата проставляються біля вершин графа (у відповідності до таблиці виходів автомата). На рис. 6.1 подана граф-схема автомата Мілі, заданого табл. 6.5.

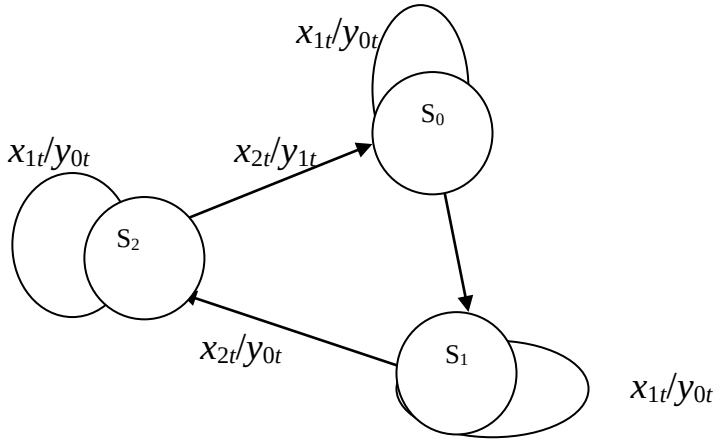


Рисунок 6.1 – Граф-схема автомата Мілі

При аналітичному способі задання абстрактні автомати подаються четвіркою об'єктів $\{X, S, Y, F\}$, де F задає для кожного стану $s_i \in S$ автомата відображення $(X \times S) \rightarrow (S \times Y)$. Іншими словами, при аналітичному способі задання для кожного стану s_i автомата вказується відображення Fs_i, яке є множиною всіх трійок s_j, x_k, y_p, причому таких, що під дією вхідного сигналу x_k автомат здійснює перехід із стану s_i в стан s_j, формуючи при цьому вихідний сигнал y_p. Загальне визначення абстрактного автомата рівнозначне опису функцій δ і λ у відповідності до виразів: $s_j = \delta(s_i, x_k)$; $y_p = \lambda(s_i, x_k)$. Відображення Fs_i записується так:

$$Fs_i = \{s_i(x_k/y_p), s_m(x_e/y_r), \ldots\}.$$

6.3. Приклади синтезу абстрактних автоматів

Приклад 1. Підкидається монетка, і робляться замітки при кожному другому підряд випадінні герба і кожному другому не обов'язково підряд випадінні цифри. Необхідно синтезувати абстрактний автомат.

Визначимо множину вхідних станів $X = \{x_0, x_1\}$, де x_0 – герб, x_1 – цифра. Визначимо множину вихідних станів $Y = \{y_0, y_1\}$, де y_0 – не робити замітку, y_1 – робити замітку. Виберемо внутрішні стани автомата. У відповідності до умови треба запам'ятати випадіння як герба, так і цифри на попередньому кроці. Оскільки автомат також має враховувати кожне друге не обов'язково підряд випадіння цифри, то треба ввести стан, який буде зберігати інформацію щодо випадіння підряд цифри і герба. Назвемо такий стан «цифра-герб». Таким чином, визначимо такі внутрішні стани: s_0 – початковий стан, s_1 – випадіння герба, s_2 – випадіння цифри, s_3 – випадіння підряд цифри і герба.

Нехай в результаті підкидання монетки, отримали таку комбінацію вхідних сигналів: г г ц ц г ц ц ц г ц.

Розглянемо реакцію автомата на дану послідовність у вигляді таблиці-стрічки (табл. 6.6).

Таблиця 6.6 – Таблиця-стрічка для прикладу 1

Стани		Вхідні сигнали									
		г	г	ц	ц	г	ц	ц	ц	г	ц
Вхідні	x_i	x_0	x_0	x_1	x_1	x_0	x_1	x_1	x_1	x_0	x_1
Внутрішні	s_i	s_1	s_0	s_2	s_0	s_1	s_2	s_0	s_2	s_3	s_0
Вихідні	y_i	y_0	y_1	y_0	y_1	y_0	y_0	y_1	y_0	y_0	y_1

Якщо в послідовності випадають два герби або дві цифри підряд, то автомат переходить в початковий стан, а на виході з'являється сигнал y_1. Якщо після випадіння герба випадає цифра, то автомат переходить у стан s_2 – цифра. Випадіння герба на попередньому кроці нас більше не цікавить. Зовсім інша ситуація відбувається у випадку випадіння цифри, а потім – герба. Інформація про це має бути збережена у вигляді стану s_3 – «цифра-герб». Якщо після цього стану випаде герб, то автомат збереже стан цифри – s_2. Якщо після цього

випаде цифра, то автомат перейде в стан s_0. І в першому, і в другому випадку на виході з'явиться вихідний стан y_1.

Керуючись викладеними вище міркуваннями, побудуємо основну таблицю абстрактного автомата для прикладу 1 (табл 6.7).

Таблиця 6.7 – Основна таблиця абстрактного автомата для прикладу 1

Стани	Входи	
	x_{1t}	x_{2t}
$s_{0(t-1)}$	s_{1t} / y_{0t}	s_{2t} / y_{0t}
$s_{1(t-1)}$	s_{0t} / y_{1t}	s_{2t} / y_{0t}
$s_{2(t-1)}$	s_{3t} / y_{0t}	s_{0t} / y_{1t}
$s_{3(t-1)}$	s_2 / y_{1t}	s_{0t} / y_{1t}

Оскільки кожному стану автомата неможливо приписати вихідний сигнал, то маємо автомат Мілі.

Побудуємо повну матрицю абстрактного автомата (табл. 6.8). Дана матриця являється частково-визначеною таблицею. Відповідно до табл. 6.6 із стану $s_{0(t-1)}$ можна потрапити тільки в стан s_{1t} або s_{2t}. Тому в клітинки на перетині стовпчиків s_{0t} і s_{3t} та рядка $s_{0(t-1)}$ вписуємо тире. У стан s_{1t} потрапляємо, подаючи на вхід x_{0t}. На виході при цьому буде сигнал y_{0t}. Отже, на перетині рядка $s_{0(t-1)}$ і стовпчика s_{1t} маємо – x_{0t} / y_{0t}. Аналогічно заповнюємо решту клітинок таблиці і будуємо граф-схему автомата (рис. 6.2).

Таблиця 6.8 – Повна матриця абстрактного автомата для прикладу 1

Стани автомата	Стани автомата s_{it}			
$s_{i(t-1)}$	s_{0t}	s_{1t}	s_{2t}	s_{3t}
$s_{0(t-1)}$	–	x_{0t} / y_{0t}	x_{1t} / y_{0t}	–
$s_{1(t-1)}$	x_{0t} / y_{1t}	–	x_{1t} / y_{0t}	–
$s_{2(t-1)}$	x_{1t} / y_{1t}	–	–	x_{0t} / y_{0t}
$s_{3(t-1)}$	x_{1t} / y_{1t}	–	x_{0t} / y_{1t}	–

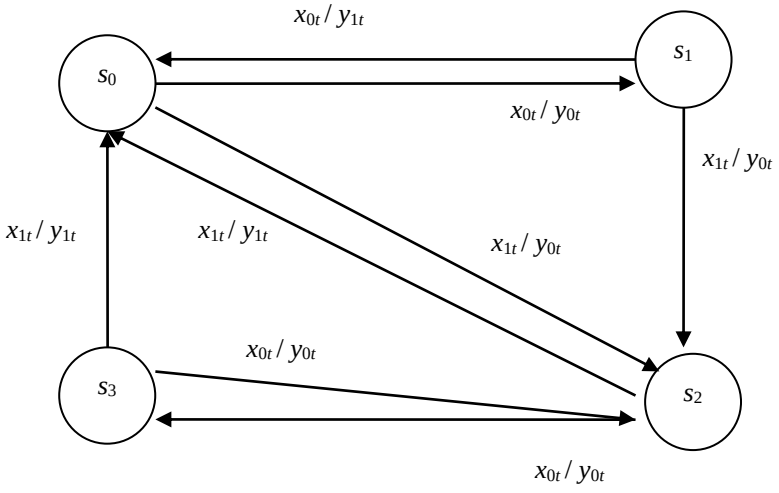


Рисунок 6.2 – Граф-схема автомата для прикладу 1

Приклад 2. Маємо англійський текст, який містить букви *a, b, c, …, z* і пробіл. Підрахувати кількість слів, які починаються на букви *un* і закінчуються на букву *d*. Синтезувати абстрактний автомат.

Визначимо множину вхідних станів *X*. Через те що на вхід надходять 27 різних сигналів, виберемо з них значущі. Для визначення роботи автомата мають значення букви *u, n, d* і пробіл. Решту букв, які надходять на вхід, можна закодувати одним станом, оскільки реакція на них буде однакова. Закодуємо вхідні стани: x_0 – поява букви *u*, x_1 – поява букви *n*, x_2 – поява букви *d*, x_3 – поява пробілу, x_4 – поява будь-якої іншої букви. Таким чином, множина вхідних станів $X = \{x_0, x_1, x_2, x_3, x_4\}$. Визначимо вихідні стани $Y = \{y_0, y_1\}$, де y_0 – не рахувати, y_1 – рахувати. Задамо внутрішні стани: s_0 – початковий стан, s_1 – поява букви *u*, s_2 – поява букв *un*, s_3 – поява букв *un…d*. Якщо автомат знаходився в стані s_0 або s_1 і на вхід поступила буква з множини x_4, то автомат переходить у стан s_4 – очікування появи пробілу.

Оскільки кожний стан автомата неможливо зв'язати з вихідним сигналом, то маємо автомат Мілі.

Будуємо основну таблицю абстрактного автомата (табл. 6.9).

Таблиця 6.9 – Основна таблиця абстрактного автомата для прикладу 2

Стани	Входи				
	x_{0t}	x_{1t}	x_{2t}	x_{3t}	x_{4t}
$s_{0(t-1)}$	s_{1t} / y_{0t}	s_{4t} / y_{0t}	s_{4t} / y_{0t}	s_{0t} / y_{0t}	s_{4t} / y_{0t}
$s_{1(t-1)}$	s_{4t} / y_{0t}	s_{2t} / y_{0t}	s_{4t} / y_{0t}	s_{0t} / y_{0t}	s_{4t} / y_{0t}
$s_{2(t-1)}$	s_{2t} / y_{0t}	s_{2t} / y_{0t}	s_{3t} / y_{0t}	s_{0t} / y_{0t}	s_{2t} / y_{0t}
$s_{3(t-1)}$	s_{2t} / y_{0t}	s_{2t} / y_{0t}	s_{3t} / y_{0t}	s_{0t} / y_{1t}	s_{2t} / y_{0t}
$s_{4(t-1)}$	s_{4t} / y_{0t}	s_{4t} / y_{0t}	s_{4t} / y_{0t}	s_{0t} / y_{0t}	s_{4t} / y_{0t}

Розглянемо роботу автомата. Якщо автомат знаходився в стані $s_{0(t-1)}$ і на його вхід була подана буква *u* (стан x_0), то він перейде в стан s_{1t}. Поява на вході будь-якої іншої букви переведе його в стан s_{4t}. Якщо автомат знаходився в стані $s_{1(t-1)}$ і на його вхід була подана буква *n* (стан x_1), то він перейде в стан s_{2t}. Поява на вході будь-якої іншої букви переведе його в стан s_{4t}. Якщо автомат знаходився в стані $s_{2(t-1)}$ і на його вхід була подана буква *d* (стан x_2), то він перейде в стан s_{3t}. Поява на вході будь-якої іншої букви залишить його в стані s_2, оскільки слово буде починатись на *un*. Якщо автомат знаходився в стані $s_{3(t-1)}$ і на його вхід був поданий пробіл (стан x_3), то він перейде в стан s_{0t} і на виході з'явиться одиничний сигнал. Поява на вході будь-якої іншої букви переведе його в стан s_2, оскільки слово все також буде починатись на *un*. Якщо автомат знаходився в стані $s_{4(t-1)}$, то поява на вході будь-якої букви залишить його в цьому стані. Тільки поява пробілу переведе автомат у стан s_{0t}.

Приклад 3. На вхід пристрою подаються цифри: 0, 1, 2. Пристрій являється суматором за модулем три. Синтезувати абстрактний автомат.

Визначимо множину вхідних станів $X = \{x_0, x_1, x_2\}$, де $x_0 - 0$, $x_1 - 1$, $x_2 - 2$. Оскільки автомат являється суматором за модулем три, то на його виході можуть з'являтись цифри: 0, 1, 2.

Визначимо вихідні стани $Y = \{y_0, y_1, y_2\}$, де $y_0 - 0$, $y_1 - 1$, $y_2 - 2$. Автомат має зберігати накопичену суму на попередньому кроці. Таким чином, визначимо такі внутрішні стани: s_0 – накопичена сума дорівнює 0, s_1 – накопичена сума дорівнює 1, s_2 – накопичена сума дорівнює 2.

Робота автомата зводиться до складання суми, накопиченої на попередньому кроці, з вхідною цифрою і передачі накопиченої суми на вихід. Складання виконується за модулем три, тобто, якщо сума дорівнює трьом, то автомат скидається в нульовий стан. Оскільки накопичена сума передається на вихід, то кожному стану автомата можна приписати вихідний сигнал. Отже, маємо автомат Мура. Для автомата Мура в основній таблиці не вказуються вихідні стани, оскільки вони ідентичні внутрішнім станам автомата.

Будуємо основну таблицю абстрактного автомата (табл. 6.10).

Таблиця 6.10 – Основна таблиця абстрактного автомата для прикладу 3

Стани	Входи		
	x_{0t}	x_{1t}	x_{2t}
$s_{0(t-1)}$	s_{0t}	s_{1t}	s_{2t}
$s_{1(t-1)}$	s_{1t}	s_{2t}	s_{0t}
$s_{2(t-1)}$	s_{2t}	s_{0t}	s_{1t}

Розглянемо роботу автомата. Подача на вхід нуля (стан x_{0t}) не змінить стан автомата. Якщо автомат знаходився в стані $s_{0(t-1)}$, то при подачі на його вхід одиничного сигналу (стан x_{1t}), він перейде в одиничний стан s_{1t}. При подачі на його вхід двійки (стан x_{2t}), він перейде в стан s_{2t}. Якщо автомат знаходився в стані $s_{1(t-1)}$, то при подачі на його вхід одиничного сигналу (x_{1t}), він перейде в стан s_{2t}. При подачі на його вхід двійки (стан x_{2t}), він скинеться в початковий стан s_{0t}. Якщо автомат знаходився в стані $s_{2(t-1)}$, то при подачі на його вхід одиничного сигналу (стан x_{1t}), він перейде в початковий стан s_{0t}. При подачі на його вхід двійки (стан x_{2t}), він перейде в стан s_{1t}.

Контрольні запитання

1. У чому відмінність абстрактної та структурної теорії автоматів?
2. Якими системами рівнянь описуються автомат Мілі та автомат Мура?
3. Назвіть і опишіть способи задання абстрактних автоматів.
4. Синтезувати абстрактний автомат для розливу кави. Автомат приймає купюри 1, 2, 5 гривень. Кава коштує 5 гривень. Автомат може видавати здачу.
5. На вхід пристрою подаються цифри 0, 1, 2. Автомат видає на вихід одиничний сигнал, якщо накопичена сума вхідних сигналів дорівнює або більше 3. Синтезувати абстрактний автомат.
6. Маємо український текст. Підрахувати кількість слів, які починаються на букву «п» та закінчуються на «ов».

7. СИНТЕЗ СТРУКТУРНОГО АВТОМАТА

7.1. Етапи канонічного методу структурного синтезу автоматів

Якщо синтез комбінаційних схем зводиться до реалізації аналітичних виразів булевих функцій за допомогою логічних елементів, то синтез цифрових автоматів з пам’яттю не настільки очевидний. У загальному випадку задача структурного синтезу автоматів з пам’яттю зводиться до знаходження загальних прийомів побудови структурних схем складних автоматів на основі композиції деяких елементарних автоматів, тобто зводиться до пошуку певних способів їх з’єднання між собою. Слід підкреслити, що далеко не при всякому виборі системи елементарних автоматів можна побудувати (шляхом їх композиції) будь-який структурний автомат. У тому випадку, коли це можливо, говорять, що задана система елементарних автоматів структурно повна. Але і на основі структурно повних систем елементарних автоматів ефективно вирішити задачу структурного синтезу довільного автомата з пам’яттю поки удається тільки для структурно повних систем елементарних автоматів деякого спеціального виду. Розглянемо один з таких методів синтезу, який дозволяє звести задачу структурного синтезу довільного автомата з пам’яттю до задачі синтезу комбінаційних схем. Метод синтезу, в основу якого покладений вказаний принцип, отримав назву канонічного методу структурного синтезу автоматів з пам’яттю. Канонічний метод структурного синтезу оперує з елементарними автоматами, які поділяються на два великих класи. Перший клас складають елементарні автомати з пам’яті, які називають елементами пам’яті. Другий клас складають елементарні комбінаційні автомати – логічні елементи. Для зведення задачі структурного синтезу довільного автомата з пам’яттю до задачі синтезу комбінаційних схем накладають обмеження на тип елементів пам’яті. Результатом роботи методу являються рівняння булевих функцій автомата в канонічній формі подання. Необхідними даними для початку роботи методу служить абстрактний цифровий автомат з пам’яттю.

Канонічний метод структурного синтезу умовно можна розділити на такі етапи:

1) кодування станів;
2) побудова канонічної таблиці структурного автомата;
3) вибір елементів пам’яті автомата;
4) побудова таблиці збудження тригерів;
5) побудова рівнянь булевих функцій виходів і збудження автомата;
6) побудова функціональної схеми автомата.

7.2. Кодування станів

Довільний цифровий автомат з пам'яттю A на абстрактному рівні подання може бути описаний у вигляді $A = \{X, Y, S, \delta, \lambda\}$. Під час переходу на структурний рівень подання кожна буква x_i вхідного алфавіту X автомата подається як двійковий вектор, тобто двійковий набір, кількість компонентів якого дорівнює кількості фізично реалізованих вхідних каналів структурного автомата. Іншими словами, кожна буква $x_i \in X$ кодується двійковим вектором. Очевидно, що мінімальна кількість необхідних фізично реалізованих вхідних каналів $k_{вх}$ в автоматі може бути підрахована за формулою $k_{вх} > \log_2|X|$, де $|X|$ – потужність алфавіту входів X. Наприклад, якщо $X = \{x_1, x_2, x_3\}$, то $k_{вх} \geq 2$. Іншими словами, кожну букву $x_i \in X$ можна закодувати двійковим вектором, який складається не менш ніж з двох компонентів, наприклад X = {00, 01, 10}. Кожна буква y_i вихідного алфавіту Y автомата також подається як двійковий вектор, кількість компонентів якого дорівнює необхідній кількості фізично реалізованих вихідних каналів автомата. За аналогією, мінімальна кількість виходів автомата $k_{вих}$ визначається за формулою $k_{вих} \geq \log_2 |Y|$, де $|Y|$ – потужність алфавіту виходів Y.

Кожна буква s_i алфавіту станів S автомата подається двійковим вектором, мінімальна кількість компонентів якого $k_{ст}$ визначається за формулою $k_{ст} \geq \log_2 |S|$. Процес заміни букв алфавітів X, Y, S абстрактного автомата двійковими векторами називається кодуванням і може бути описаний таблицями кодування. Таблиця кодування будується так: в лівій її частині вказуються всі букви (наприклад, вхідного алфавіту абстрактного автомата), а в правій – двійкові вектори, які ставляться у відповідність до цих букв.

Розглянемо кодування станів для прикладу 1.

Приклад 1. Автомат заданий основною таблицею абстрактного автомата (табл. 7.1). Синтезувати структурний автомат.

Таблиця 7.1 – Основна таблиця абстрактного автомата

Внутрішні стани	Вхідні стани	
	x_{0t}	x_{1t}
$s_{0\,(t\text{-}1)}$	$s_{1\,t} / y_{0\,t}$	$s_{1\,t} / y_{0\,t}$
$s_{1\,(t\text{-}1)}$	$s_{2\,t} / y_{0\,t}$	$s_{0\,t} / y_{1\,t}$
$s_{2\,(t\text{-}1)}$	$s_{3\,t} / y_{0\,t}$	$s_{2\,t} / y_{0\,t}$
$s_{3\,(t\text{-}1)}$	$s_{0\,t} / y_{1\,t}$	$s_{3\,t} / y_{0\,t}$

Кодуємо стани автомата. Позначимо буквою A вхідний алфавіт, буквою B – алфавіт внутрішніх станів, буквою C – вихідний алфавіт. Визначаємо кількість бітів $k_{вх}$, необхідних для кодування вхідних станів, як $k_{вх} \geq \log_2 X$, де X – кількість станів. Кількість вхідних станів $X = 2$, тому кількість бітів, необхідних для кодування вхідних станів $k_{вх} = \log_2 2 = 1$. Складаємо таблицю кодування вхідних станів (табл.. 7.2). Визначаємо кількість бітів $k_{ст}$, необхідних

для кодування внутрішніх станів, як $k_{ст} = \log_2 4 = 2$. Складаємо таблицю кодування внутрішніх станів (табл.. 7.3). Визначаємо кількість бітів $k_{вих}$, необхідних для кодування вихідних станів: $k_{вих} = \log_2 2 = 1$. Складаємо таблицю кодування вихідних станів (табл.. 7.4).

Таблиця 7.2 – Кодування вхідних станів

Стан	Код
x_{0t}	0
x_{1t}	1

Таблиця 7.3 – Кодування внутрішніх станів

Стан	Код
$s_{0\,(t\text{-}1)}$	00
$s_{1\,(t\text{-}1)}$	01
$s_{2\,(t\text{-}1)}$	10
$s_{3\,(t\text{-}1)}$	11

Таблица 7.4 – Кодування вихідних станів

Стан	Код
$y_{0\,t}$	0
$y_{1\,t}$	1

7.3. Побудова канонічної таблиці структурного автомата

Для забезпечення зручності побудови канонічної таблиці структурного автомата в прикладі 7.1, наприклад, перетворимо основну таблицю абстрактного автомата, замінивши в ній позначення станів на їх коди (табл. 7.5). Будуємо канонічну таблицю структурного автомата (табл. 7.6). Ліва частина таблиці подає двійкові коди алфавіту вхідних станів в поточний момент і внутрішніх станів в попередній момент часу. Права частина таблиці містить коди алфавіту стану автомата і вихідні стани автомата в поточний момент часу. Розглянемо заповнення першого рядка табл. 7.6. В табл. 7.5 під час подачі на вхід 0, знаходячись в стані 00, автомат перейде в стан 01 і на виході буде 0. Отже, права частина першого рядка табл. 7.6 буде мати вигляд 010. Наступні рядки заповнюються аналогічно.

Таблиця 7.5 – Закодована таблиця абстрактного автомата

Коди внутрішніх станів в момент (t-1)	Коди вхідних станів	
	0	1
00	01/0	01/0
01	10/0	00/1
10	11/0	10/0
11	00/1	11/0

Таблиця 7.6 – Канонічна таблиця структурного автомата

A_t	$B_{1(t\text{-}1)}$	$B_{2(t\text{-}1)}$	B_{1t}	B_{2t}	C_t
0	0	0	0	1	0
0	0	1	1	0	0
0	1	0	1	1	0
0	1	1	0	0	1
1	0	0	0	1	0
1	0	1	0	0	1
1	1	0	1	0	0
1	1	1	1	1	0

7.4. Вибір елементів пам’яті автомата

Заміна таблиці переходів автомата на структурну таблицю переходів призводить до того, що функція переходів δ автомата стає векторною. Іншими словами, аргументами такої функції є всі можливі пари двійкових векторів (s_i, x_i), а сама функція набуває значення із множини S двійкових векторів станів автомата. У відповідності до структурної таблиці переходів автомата його векторна функція переходів кожній парі двійкових векторів (s_i, x_i) ставить у відповідність визначений двійковий вектор s_k, що на абстрактному рівні визначається співвідношенням $s_k = \delta\,(s_i, x_i)$. З цього випливає, що структурний автомат повинен запам’ятовувати двійковий вектор кожного наступного стану автомата, для чого і служать елементи пам’яті.

При канонічному методі структурного синтезу автоматів як елементи пам’яті використовуються елементарні автомати Мура з двома станами, які мають повну систему переходів і виходів.

Як елементи пам’яті структурного автомата звичайно використовуються *D*-тригери, *T*-тригери, *RS*-тригери, *JK*-тригери, які задовольняють вимоги відносно повноти переходів і виходів (рис. 7.1).

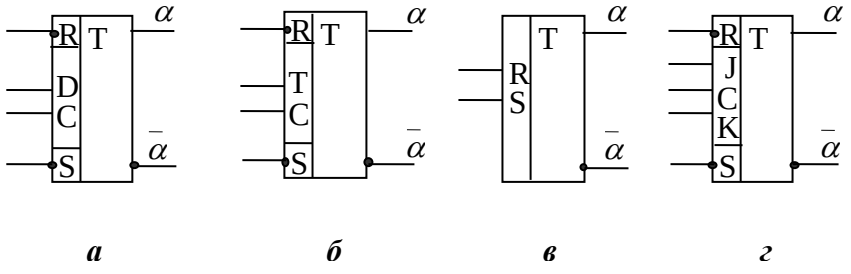


Рисунок 7.1 – Елементи пам’яті автомата:
а – *D*-тригер; *б* –*T*-тригер; *в* – *RS*-тригер; *г* – *JK*-тригер

Таблиці переходів вказаних тригерів подані табл. 7.7, 7.8, 7.9 і 7.10 відповідно.

Таблиця 7.7 – Таблиця переходів *D*-тригера

D_t	$\alpha_{(t\text{-}1)}$	α_t
0	0	0
0	1	0
1	0	1
1	1	1

Таблиця 7.8 – Таблиця переходів *T*-тригера

T_t	$\alpha_{(t\text{-}1)}$	α_t
0	0	0
0	1	1
1	0	1
1	1	0

Таблиця 7.9 – Таблиця переходів *RS*-тригера

R_t	S_t	$\alpha_{(t\text{-}1)}$	α_t
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	*	*
1	1	*	*

Таблиця 7.10 – Таблиця переходів *JK*-тригера

J_t	K_t	$\alpha_{(t\text{-}1)}$	α_t
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Входи *D*, *T*, *R*, *S*, *J* і *K* називаються інформаційними. Таблиці переходів тригерів складаються тільки для інформаційних входів. Інші входи являються допоміжними. Зокрема, у *RS*-тригера вхід *R* – вхід установки тригера в нуль,

вхід S – вхід установки тригера в одиницю, а вхід C – вхід для підключення синхроімпульсу. Кожний з тригерів має два виходи. Поява одиничного сигналу на виході, який позначений на рисунках символом α, означає, що тригер перейшов у одиничний стан. Поява одиничного сигналу на виході $\bar{\alpha}$ означає, що тригер перейшов у нульовий стан. Для *RS*-тригера являється забороненою вхідна комбінація $RS = 11$, тому що може призвести до неоднозначної роботи тригера. Кожний з вказаних тригерів є автоматом Мура з повною системою переходів (останнє видно із таблиць переходів тригерів) і повною системою виходів (кожний стан тригера являється його вихідним сигналом).

7.5. Побудова таблиці збудження тригера

Побудуємо таблиці збудження *T*-тригера і *D*-тригера для прикладу 1. Далі наведені пояснення щодо *T*-тригера.

Оскільки в прикладі 7.1 внутрішні стани автомата кодуються за допомогою двох бітів (табл.7.3), то для структурної схеми цього автомата знадобляться два *T*-тригери – T_1 і T_2.

Ліва частина таблиці збудження тригера містить двійкові коди алфавіту вхідних станів у поточний момент і станів тригера в попередній момент часу. Права частина таблиці містить стани тригера в поточний момент часу. Таблиця збудження тригера будується за канонічною таблицею структурного автомата так. Розглянемо перший рядок табл. 7.6. Стан $B_{1(t\text{-}1)} = 0$ зіставляємо з станом тригера $T_{1(t\text{-}1)} = 0$ першого рядка табл. 7.11. Стан $B_{1(t\text{-}1)}$ (тобто $T_{1(t\text{-}1)}$) у відповідності до табл.7.6 треба змінити на стан $B_{1t} = 0$ (тобто $T_{1t} = 0$). Із таблиці переходів *T*-тригера (табл.7.8) маємо: якщо тригер був у стані $\alpha_{(t\text{-}1)} = 0$ і йому необхідно перейти в стан $\alpha_t = 0$, то на вхід тригера T_t треба подати нульовий сигнал. Отже, в клітинці першого рядка і стовпця T_{1t} табл. 7.11 пишемо 0. Стан $B_{2t\text{-}1} = 0$ в першому рядку змінюється на стан $B_{2t} = 1$. Отже, у відповідності до табл. 7.8 на вхід тригера тепер необхідно подати 1. Таким чином, в клітинку табл. 7.11, розташовану на перетині першого рядка і стовпчика T_{2t}, вписуємо 1. Решту рядків треба заповнити таким же чином.

Таблиця 7.11 – Таблиця збудження *T*-тригера

A_t	$T_{1(t\text{-}1)}$	$T_{2(t\text{-}1)}$	T_{1t}	T_{2t}
0	0	0	0	1
0	0	1	1	1
0	1	0	0	1
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	0	0

Аналогічно будуємо таблицю збудження *D*-тригера (табл. 7.12).

Таблиця 7.12 – Таблиця збудження *D*-тригера

A_t	$T_{1(t\text{-}1)}$	$T_{2(t\text{-}1)}$	T_{1t}	T_{2t}
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	0	0	0	1
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

Якщо отриману таблицю збудження *D*-тригера зіставити з канонічною таблицею структурного автомата для прикладу 7.1 (табл. 7.6), то ми побачимо, що вони ідентичні.

7.6. Побудова рівнянь функцій збудження і виходів автомата

Побудуємо систему рівнянь збудження *T*-тригерів і систему рівнянь виходів. За функціонально повний базис виберемо базис ***АБО-НІ*** (стрілка Пірса).

Переносимо значення функції збудження T_{1t} із табл. 7.11 в карту Карно (рис. 7.2).

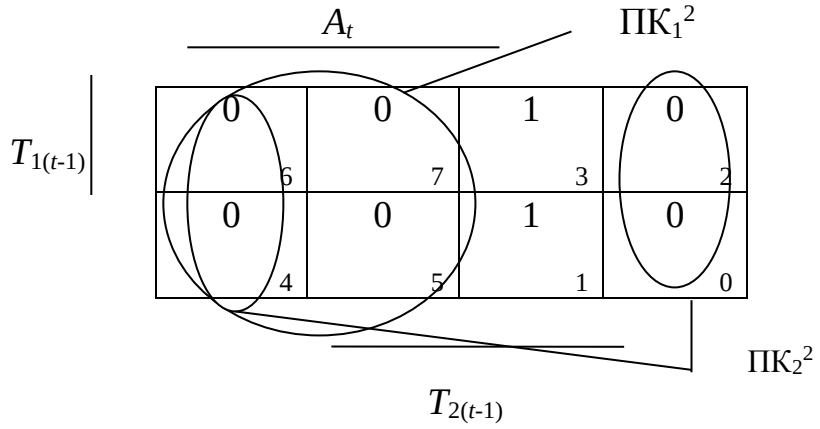


Рисунок 7.2 – Карта Карно для функції збудження T_{1t}

Оскільки реалізувати схему необхідно в базисі ***АБО-НІ***, то будуємо правильні конфігурації за нульовими наборами. В ПК$_1^2$ об’єднуємо клітинки 4, 5, 6 і 7. В ПК$_2^2$ об’єднуємо клітинки 0, 2, 4 і 6. Описуємо ПК за допомогою табл. 7.13.

Таблиця 7.13 – Правильні конфігурації для функції збудження T_{1t}

Правильні конфігурації	Значення змінних		
	A_t	$T_{1(t\text{-}1)}$	$T_{2(t\text{-}1)}$
$ПК_1^2$	1	–	–
$ПК_2^2$	–	–	0

Складаємо рівняння збудження першого *T*-тригера. Хоча в базисі ***АБО-НІ*** змінні, які описують ПК, беруться в інверсному вигляді, тут мають місце виключні ситуації для першої і другої правильних конфігурацій.

$$T_{1t} = A_t \downarrow \overline{T_{2(t-1)}} \qquad (7.1)$$

Заносимо в карту Карно значення функції збудження T_{2t} (рис. 7.3).

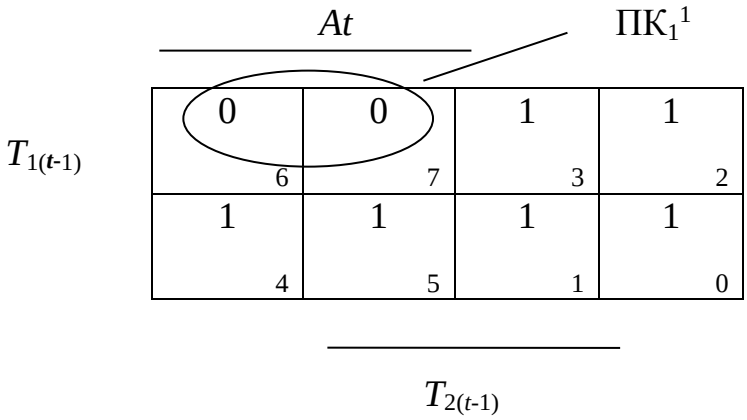


Рисунок 7.3 – Карта Карно для функції збудження T_{2t}

Через те що реалізувати схему необхідно в базисі ***АБО-НІ***, то будуємо правильні конфігурації за нульовими наборами. В $ПК_1^1$ об'єднуємо клітинки 6 і 7. Описуємо ПК за допомогою табл. 7.14.

Таблиця 7.14 – Правильні конфігурації для функції збудження T_{2t}

Правильні конфігурації	Значення змінних		
	A_t	$T_{1(t\text{-}1)}$	$T_{2(t\text{-}1)}$
$ПК_1^1$	1	1	–

Складаємо рівняння збудження другого *T*-тригера:

$$T_{2t} = \overline{\overline{A_t} \downarrow \overline{T_{1(t-1)}}} \qquad (7.2)$$

Будуємо системи рівнянь для булевих функцій виходів. Значення C_t з табл. 7.6 заносимо в карту Карно (рис. 7.4), замінюючи при цьому $B_{1(t-1)}$ на $T_{1(t-1)}$ і $B_{2(t-1)}$ на $T_{2(t-1)}$.

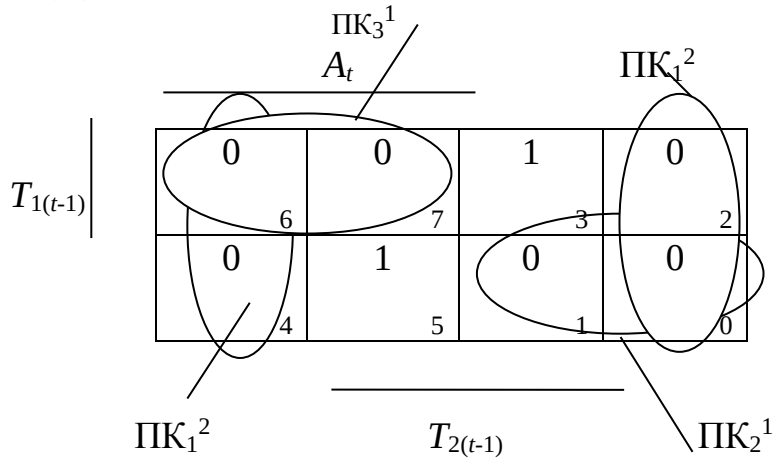


Рисунок 7.4 – Карта Карно для функції виходу C_t

Оскільки реалізувати схему необхідно в базисі ***АБО-НІ***, то будуємо правильні конфігурації за нульовими наборами. В $ПК_1^2$ об'єднуємо клітинки 0, 2, 4 і 6. В $ПК_2^1$ об'єднуємо клітинки 0 і 1. В $ПК_3^1$ об'єднуємо клітинки 6 і 7. Описуємо ПК за допомогою табл. 7.14.

Таблиця 7.14 – Правильні конфігурації для вихідного стану C_t

Правильні конфігурації	Значення змінних		
	A_t	$T_{1(t-1)}$	$T_{2(t-1)}$
$ПК_1^2$	–	–	0
$ПК_2^1$	0	0	–
$ПК_3^1$	1	1	–

Складаємо рівняння для вихідного сигналу C_t. Хоча в базисі ***АБО-НІ*** змінні, які описують ПК, беруться в інверсному вигляді, в даному випадку має місце виключна ситуація для першої правильної конфігурації. Тому

$$C_t = \overline{T_{2(t-1)}} \downarrow (A_t \downarrow T_{1(t-1)}) \downarrow (\overline{A_t} \downarrow \overline{T_{1(t-1)}}). \qquad (7.3)$$

7.7. Побудова функціональної схеми автомата

Функціональна схема автомата наведена на рис. 7.7.

Введемо цифрові позначення: 1 – прямий вхідний сигнал A_t; 2 – інверсний вхідний сигнал $\overline{A_t}$; 3 – прямий вихід тригера T_1; 4 – інверсний вихід тригера T_1; 5 – прямий вихід тригера T_2; 6 – інверсний вихід тригера T_2; 7 – вихідний сигнал C_t. Згідно з рівнянням (7.1) на вхід тригера T_1 має подаватись сигнал з

виходу схеми ***АБО-НІ***, на вхід якої треба подати вхідний сигнал A_t і інверсний вихідний сигнал другого тригера T_2.

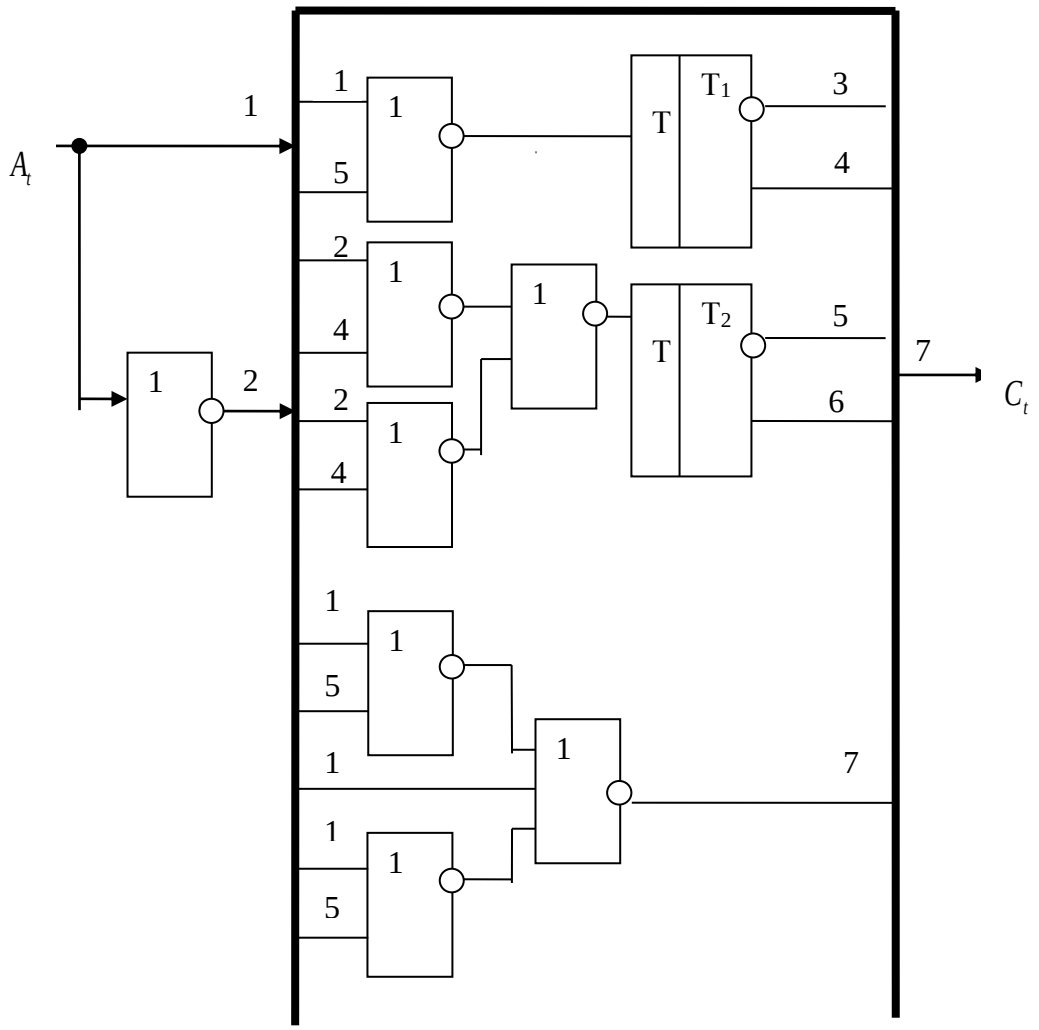


Рисунок 7.5 – Функціональна схема автомата

Аналогічно треба побудувати іншу частину схеми у відповідності до рівнянь (7.2) і (7.3).

Контрольні запитання

1. Назвіть два класи елементарних автоматів.
2. Які етапи канонічного методу структурного синтезу автомата ви знаєте?
3. Як будується основна таблиця абстрактного автомата?
4. Як кодуються вхідні, внутрішні і вихідні стани автомата?
5. Як будується закодована таблиця структурного автомата?
6. Як будується канонічна таблиця структурного автомата?

7. Які тригери використовуються в якості елементів пам’яті структурного автомата?
8. Які особливості побудови таблиць переходів для різних тригерів?
9. Як будується таблиця збудження тригера?
10. Як складаються системи рівнянь функцій збудження тригера і виходів автомата?
11. Як описуються правильні конфігурації карт Карно?
12. Як побудувати функціональну схему автомата?

8. ПРОЕКТУВАННЯ КОМБІНАЦІЙНИХ СХЕМ НА ДЕШИФРАТОРАХ І МУЛЬТИПЛЕКСОРАХ

8.1. Синтез схем на дешифраторах

Дешифратором називається комбінаційна схема, яка реалізує всі конституенти одиниці. Іншими словами, дешифратор – це схема, що має n входів і 2^n виходів. Вона включає один із вихідних каналів, тобто тільки на одному з виходів з'являється одиничне значення, при подачі відповідного двійкового набору на n входів. Дешифратор являється типовою комбінаційною схемою, яка знаходить широке застосування в обчислювальній техніці. Тому дешифратори на невелику кількість змінних, а саме, коли n не більше чотирьох, виготовляються у вигляді стандартних мікросхем. Однак, в ряді випадків на практиці приходиться будувати дешифратори з десятками тисяч виходів, наприклад, в запам'ятовуючих пристроях і електронних комутаторах. Дешифратор, який має три входи, задається таблицею істинності, що наведена в табл. 8.1.

Таблиця 8.1 – Таблиця істинності дешифратора на три входи

x_1	x_2	x_3	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Із визначення конституенти одиниці очевидно, що під час синтезу дешифраторів доцільно використовувати тільки елементи типу ***І*** або ***АБО-НІ***. Умовне позначення дешифратора на три входи подане на рис. 8.1

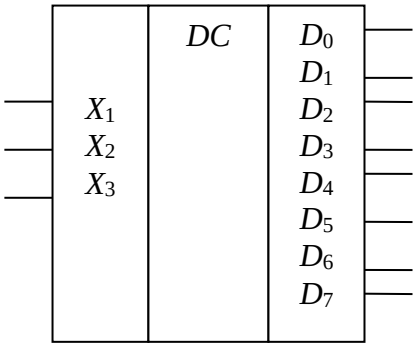


Рисунок 8.1 – Дешифратор на три входи

За допомогою дешифратора легко реалізувати довільну булеву функцію, об’єднавши за допомогою схеми ***АБО*** ті його виходи, які відповідають одиницям в таблиці істинності булевої функції. Об’єднувати можна і виходи, які відповідають нулям в таблиці істинності булевої функції. Але для об’єднання необхідно використати схему ***АБО-НІ***.

Приклад 1. Реалізувати за допомогою дешифратора суматор за модулем два («Виключне АБО»).

Таблиця істинності суматора за модулем два має такий вигляд (табл. 8.2):

Таблиця 8.2 – Таблиця істинності суматора за модулем два

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Нам знадобиться дешифратор на два входи. Якщо ми об’єднаємо за допомогою схеми ***АБО*** ті його виходи, які відповідають одиницям в табл. 8.2 (набори 01 і 10), то отримаємо рівняння (8.1).

$$y = D_1 \vee D_2 \tag{8.1}$$

Реалізація суматора за модулем два за допомогою дешифратора наведена на рис. 8.2.

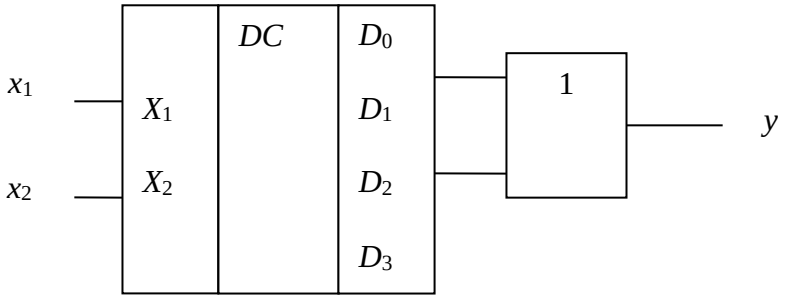


Рисунок 8.2 – Реалізація суматора за модулем два за допомогою дешифратора

Приклад 2. Реалізувати за допомогою дешифратора пристрій візуальної індикації десяткових цифр на світловому табло.

Позначимо змінними a_1, a_2, …, a_7 лінії індикації, за допомогою яких можна отримати зображення десяткових цифр від 0 до 9 (рис. 8.3). Так, для отримання зображення цифри нуль мають бути обраними усі лінії, окрім лінії a_7. Для отримання зображення одиниці мають бути обраними лінії a_2 і a_3. Подібні міркування для отримання зображення всіх десяткових цифр відобразимо у вигляді табл. 8.3.

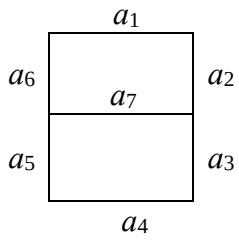


Рисунок 8.3 – Розміщення ліній візуальної індикації для отримання зображення десяткових цифр на світловому табло

Таблиця 8.3 – Таблиця отримання візуального відображення десяткових цифр

Десяткова цифра	Двійковий код				Значення ліній індикації						
	x_1	x_2	x_3	x_4	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

Нам знадобиться дешифратор на чотири входи. В табл. 8.3 маємо сім вихідних функцій a_1, a_2, ..., a_7, які відповідають вибору тієї чи іншої лінії індикації. Необхідно спочатку скласти рівняння для кожної з цих функцій. Так, якщо об'єднати за допомогою схеми ***АБО-НІ*** виходи дешифратора D_1 і D_4, які відповідають нулям в таблиці істинності для вихідної функції a_1 (набори 0001 і 0100), то отримаємо реалізацію для функції a_1:

$$a_1 = \overline{D_1 \vee D_4} \tag{8.2}$$

Об'єднавши за допомогою схеми ***АБО-НІ*** виходи дешифратора D_5 і D_6, отримаємо реалізацію для функції a_2:

$$a_2 = \overline{D_5 \vee D_6} \tag{8.3}$$

Аналогічно отримаємо наступні рівняння:

$$a_3 = D_2 \tag{8.4}$$

$$a_4 = \overline{D_1 \vee D_4 \vee D_7} \tag{8.5}$$

$$a_6 = \overline{D_1 \vee D_2 \vee D_3 \vee D_7} \tag{8.6}$$

$$a_7 = \overline{D_0 \vee D_1 \vee D_7} \tag{8.7}$$

Функцію a_5 простіше реалізувати, об'єднавши за допомогою схеми ***АБО*** ті виходи дешифратора, на яких ця функція приймає одиничні значення:

$$a_5 = D_0 \vee D_2 \vee D_6 \vee D_8 \tag{8.8}$$

Реалізація пристрою візуальної індикації десяткових цифр на світловому табло за допомогою дешифратора подана на рис. 8.4.

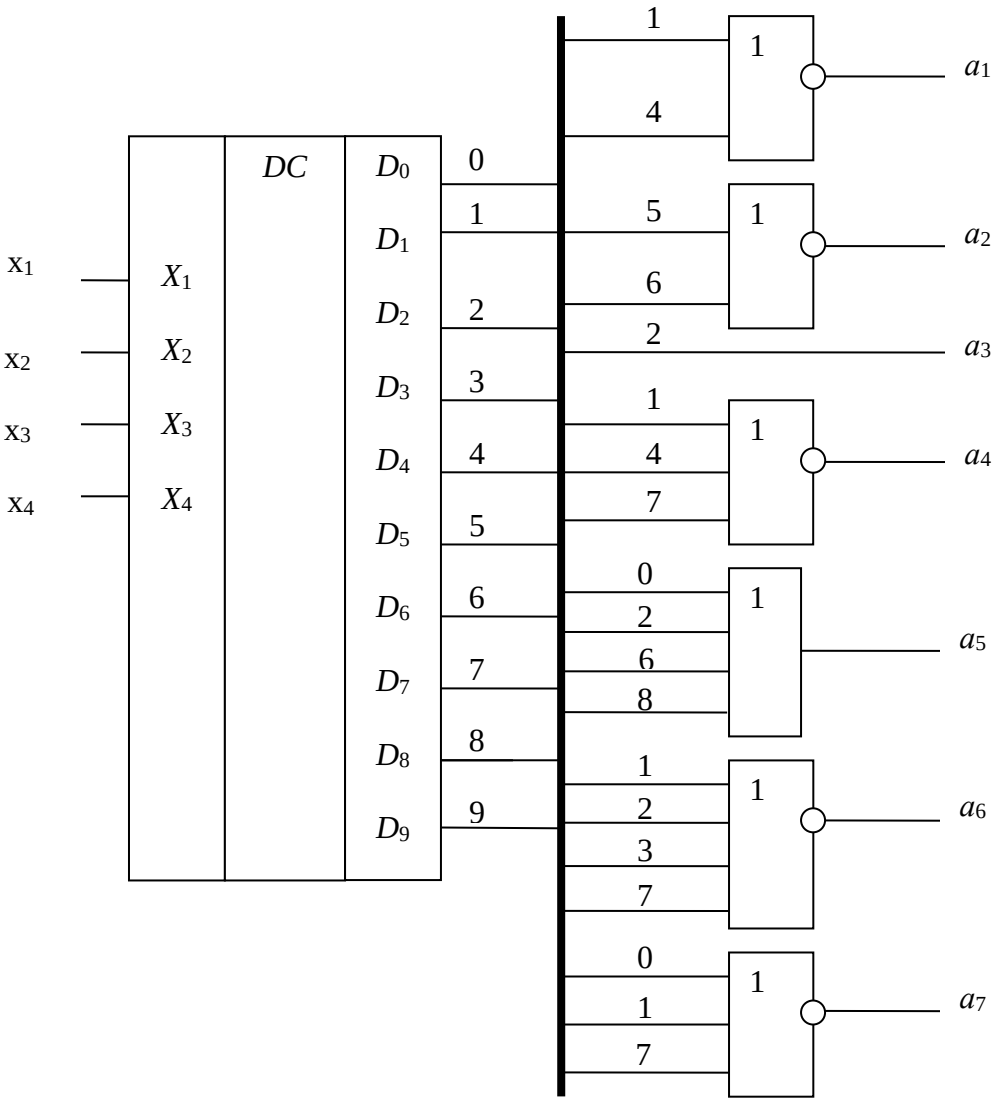


Рисунок 8.4 – Схема пристрою візуальної індикації десяткових цифр на світловому табло

8.2. Синтез схем на мультиплексорах

Мультиплексором (МП) називається конструктивний елемент з одним виходом і двома групами входів, а саме адресними входами (входи управління, інакше селекторні входи) і входами даних. МП дозволяє подачею певних двійкових векторів на адресні входи підключати до виходу потрібні вхідні дані. Фактично МП являється комутатором, який відповідний вхід даних підключає до свого виходу. Необхідний вхід даних вибирається за допомогою відповідного значення на адресних входах. Нині під час проектування засобів обчислювальної техніки широко використовуються мультиплексори, які фізично реалізовані в одному корпусі інтегральної мікросхеми середнього ступеня інтеграції.

Позначимо буквами $x_i \in \{0, 1\}$ адресні входи МП, а буквами $a_i \in \{0, 1\}$ – інформаційні входи. Якщо МП має n адресних входів, то інформаційних входів (тобто входів даних) – 2^n.

В табл. 8.4 подана таблиця істинності мультиплексора з двома адресними входами x_1 і x_2 і чотирма інформаційними входами a_0, a_1, a_2, і a_3.

Таблиця 8.4 – Таблиця істинності мультиплексора

Значення адресних входів		Значення виходу
x_1	x_2	f
0	0	a_0
0	1	a_1
1	0	a_2
1	1	a_3

Якщо на адресні входи мультиплексора подати 00, то до виходу підключиться вхідний сигнал a_0. В залежності від значення a_0 на виході з'явиться нульовий або одиничний сигнал.

Умовне зображення МП з двома адресними і чотирма інформаційними входами подане на рис. 8.5.

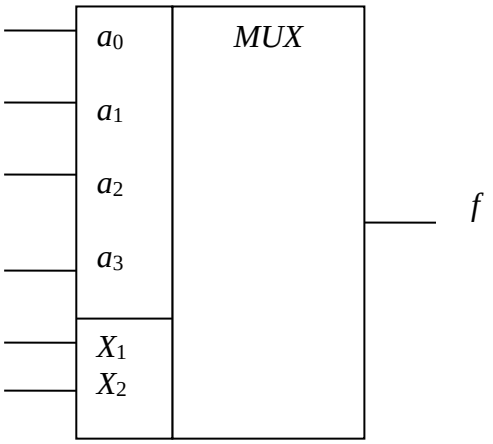


Рисунок 8.5 – Умовне зображення МП з двома адресними і чотирма інформаційними входами

Якщо на входи даних a_0, a_1, …, $a_{2\wedge k-1}$ подати двійковий вектор, який відповідає стовпцю значень заданої функції f в таблиці істинності, а на адресні входи – значення змінних, наведених в таблиці істинності, то МП реалізує функцію f (x_1, x_2, …, x_k). Відмітимо, що в описаному способі реалізації функції f на входи даних МП подаються константи (константа 0 і константа 1) у відповідності зі значеннями функції в її таблиці істинності. Приклад реалізації функції f_1 (x_1, x_2, x_3) (табл. 8.5) наведений на рис. 8.6.

В ряді випадків МП має додатковий вхід x_R, який називається стробом. Функціональне призначення строба таке: якщо $x_R = 1$, то на виході МП буде

нульове значення (незалежно від двійкового набору, поданого на адресні входи); якщо $x_R = 0$, то МП функціонуватиме як завжди.

Найчастіше використовуються МП, реалізовані у вигляді однієї інтегральної мікросхеми.

Таблиця 8.5 – Таблиця істинності функції f_1 (x_1, x_2, x_3)

x_1	x_2	x_3	f_1 (x_1, x_2, x_3)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

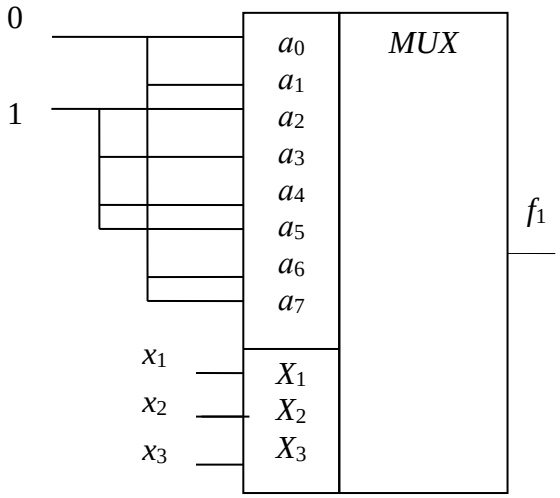


Рисунок 8.6 – Реалізація функції f_1 (x_1, x_2, x_3) на мультиплексорі

Якщо МП має n адресних входів, то на такому МП можна реалізувати будь-яку булеву функцію (n + 1) змінних. Дійсно, задамо функцію f (x_1, x_2, ..., x_{n+1}) її таблицею істинності. Змінні x_1, x_2, ..., x_n виділимо як адресні змінні МП, а змінну x_{n+1} – як змінну даних МП. Тоді можливі такі чотири ситуації для будь-яких двох сусідніх двійкових наборів довжини (n + 1) з однаковою адресною частиною: 1) $f = 1$; 2) $f = 0$; 3) $f = x_{n+1}$; 4) $f = \bar{x}_{n+1}$. Таким чином, якщо змінні x_1, x_2, ..., x_n подати на адресні входи МП, то для реалізації функції f (x_1, x_2, …, x_{n+1}) необхідно на кожному вході даних МП реалізувати або константу 0, або константу 1, або x_{n+1}, або $\bar{x}_{n+1}$ (що визначається таблицею істинності функції).

Приклад 3. Реалізуємо на МП з трьома адресними входами булеву функцію f_2 (x_1, x_2, x_3, x_4) (табл. 8.6). Виділимо змінні x_1, x_2 і x_3 як адресні змінні МП. Тоді: якщо на адресні входи МП (табл. 8.6) поступає набір 000, то $f_2 = x_4$; якщо поступають набори 001, або 010, або 100, то $f_2 = \overline{x_4}$; якщо поступають набори 011, або 101, то $f_2 = 1$ і, нарешті, якщо поступають набори 110, або 111, то $f_2 = 0$. Таким чином, отримуємо реалізацію булевої функції, подану на рис. 8.7. Константа 0 реалізується підключенням відповідного входу даних МП до "землі", а константа 1 – підключенням відповідного входу даних МП до шини живлення.

Таблиця 8.6 – Таблиця істинності функції f_2 (x_1, x_2, x_3, x_4)

Входи *MUX*	x_1	x_2	x_3	x_4	f_2	Значення на входах *MUX*
0	0	0	0	0	0	x_4
	0	0	0	1	1	
1	0	0	1	0	1	$\overline{x_4}$
	0	0	1	1	0	
2	0	1	0	0	1	$\overline{x_4}$
	0	1	0	1	0	
3	0	1	1	0	1	1
	0	1	1	1	1	
4	1	0	0	0	1	$\overline{x_4}$
	1	0	0	1	0	
5	1	0	1	0	1	1
	1	0	1	1	1	
6	1	1	0	0	0	0
	1	1	0	1	0	
7	1	1	1	0	0	0
	1	1	1	1	0	

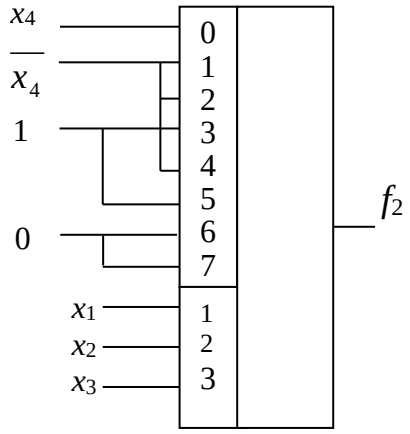


Рисунок 8.7 – Реалізація булевої функції f_2 (x_1, x_2, x_3, x_4)

Контрольні запитання

1. Призначення та особливості функціонування дешифраторів і мультиплексорів.
2. Як за допомогою дешифратора реалізувати функцію, задану її таблицею істинності, наприклад?
3. Як функціонує схема, наведена на рис. 8.4?
4. Які групи входів існують у мультиплексора?
5. Як на мультиплексорі реалізувати функцію, яка залежить від більшої кількості змінних, ніж кількість адресних входів у мультиплексора?

9. СИНТЕЗ МІКРОПРОГРАМНОГО АВТОМАТА ЗА СХЕМОЮ АЛГОРИТМУ

9.1. Послідовність дій, необхідних для побудови управляючого пристрою

Найчастіше пристрої складаються з двох частин: виконавчої та управляючої. Виконавча частина пристрою зазвичай складна та незмінна. Управляюча частина значно простіша, може змінюватись, і вся гнучкість пристрою полягає саме в гнучкості управляючої частини. В обчислювальній машині одна із складних і незмінних частин – арифметичний пристрій. Однак на ньому можна виконати як додавання, так і обчислення синуса кута, або розв'язання диференціального рівняння, або щось інше. Що саме буде отримано, залежить від управляючої частини.

Взаємодія управляючої і виконуючої частин показана на рис. 9.1.

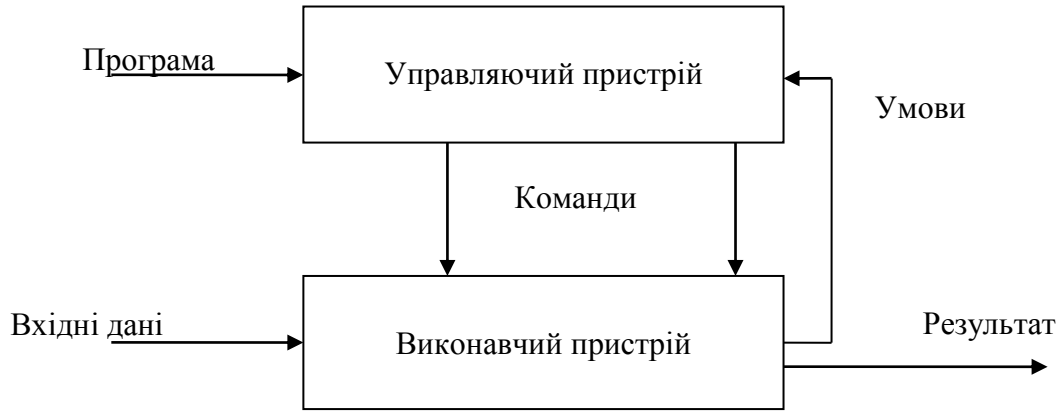


Рисунок 9.1 – Взаємодія управляючого і виконавчого пристроїв

Управляючий пристрій подає на виконавчий пристрій команди. Виконавчий пристрій виконує команди і, крім того, має набір булевих функцій, які виробляють «умови», що свідчать про ті або інші події у виконавчому пристрої. Основний потік вхідних даних надходить тільки у виконавчий пристрій. В управляючий пристрій надходить програма. Результати також надходять на вихід, минаючи управляючий пристрій. Програма завантажується в управляючий пристрій до початку обробки вхідних даних. Управляючий пристрій із завантаженою програмою є *операційним автоматом*, який настроєний на розв'язання завдання з обробки даних.

Зміст вироблення умов такий: програма повинна працювати з різними вхідними даними (тобто досить запустити її один раз). Хід обчислень може змінюватись залежно від вхідних даних. Розпізнавання ситуації, критичної для виконання програми, може бути складним завданням, тому воно виконується у виконавчому пристрої. Повідомлення про те, виникла критична ситуація чи ні, має тільки два значення: «так» або «ні». Повідомлення надходить в управляючий пристрій з виходу «Умови» виконавчого пристрою.

Таким чином, оскільки виконавчий пристрій не змінюється, то і функції, які виробляють умови, не змінюються. У роботі управляючого пристрою можуть використовуватися тільки такі умови, вироблення яких передбачене у виконавчому пристрої. Розглянемо випадок, коли потрібно виконувати ту саму програму для різних вхідних даних. Що являє собою управляючий пристрій? Як його можна реалізувати? Виявляється, відповідь можна дати мовою кінцевих автоматів. Управляючий пристрій – це кінцевий автомат. Вхідні сигнали для нього – умови і синхроімпульси, а вихідні – команди.

Створення управляючого автомата починається зі схеми алгоритму розв'язання завдання (рис. 9.2) Вона складається з таких блоків: «**Початок»,** «**Кінець»**, «**Операційна вершина»** (рис. 9.3), «**Умовна вершина**» (рис. 9.4)**.**

Початок означає початок роботи алгоритму. **Кінець** – завершення його роботи. **Операційна вершина** означає роботу, яка виконується виконавчим пристроєм у фрагменті алгоритму. Вона має виконуватися однаково для всіх допустимих вхідних даних, тобто не містити розгалужень.

Умовна вершина містить булеву змінну, значення якої забезпечує вибір одного із двох напрямків подальшої роботи алгоритму.

Стрілки на лініях відображають порядок, в якому мають виконуватись блоки в схемі.

Вміст операційних блоків може повторюватись так само, як і умови в умовних блоках.

При створенні автомата потрібно заздалегідь передбачити сукупність букв вхідного алфавіту для умовних блоків. Він має складатись із вхідних букв X_i, де x_0 – синхроімпульс. Остаточно вхідний алфавіт буде сформований у результаті розробки таблиці переходів.

Вихідний алфавіт – це команди, які формує управляючий автомат. Кожна команда запускає виконання деякого операційного блока або викликає перехід на умовний блок (без виконання операційного блока). «Порожню команду», що не запускає жодного блока, позначимо як y_0 (можна використовувати позначення у вигляді прочерку – «–»). Інші команди позначимо буквою Y_j з тим же номером, що і операційний блок, який ця команда запускає. В результаті отримаємо множину виходів.

Кількість внутрішніх станів S_k залежить від того, якого типу автомат створюється –Мілі чи Мура.

Побудова мікропрограмного автомата за схемою алгоритму (його іноді називають автоматом Уілкса-Стрінджера) зводиться до таких дій:

1) кодування операційних та умовних вершин;

2) побудова основної таблиці та граф-схеми переходів;

3) побудова системи рівнянь для функцій переходів і системи рівнянь для функцій виходів (останнє тільки для автоматів Мілі);

4) кодування внутрішніх станів автомата;

5) побудова функціональної схеми автомата.

Спочатку розглянемо послідовність дій, необхідних для побудови управляючого пристрою для автомата Мілі.

9.2. Синтез автомата Мілі

Розглянемо синтез автомата Мілі на прикладі 1.

Приклад 1. Побудувати операційний автомат, що обчислює кількість парних елементів у двох одновимірних масивах ($A[n]$ і $B[m]$). Мікропрограмний автомат реалізувати за схемою алгоритму у вигляді автомата Мілі. Функціональну схему автомата синтезувати на елементах І, АБО, НІ та *RS*-тригерах, доповнюючи її необхідними за алгоритмом функціональними автоматами.

Побудова операційного автомата зводиться до описаних нижче дій.

9.2.1. Побудова змістовної схеми алгоритму. До складу схеми алгоритму (рис. 9.2) входять операційні (рис. 9.3) та умовні вершини (рис. 9.4). Запропонований алгоритм виконує знаходження кількості парних елементів у двох одновимірних масивів розмірності $[n]$ і $[m]$ відповідно. При цьому використовуються чотири умовні вершини і десять операційних. Підрахунок сумарної кількості парних елементів в обох масивах виконується поступово.

9.2.2.Побудова таблиці кодування операційних та умовних вершин. Кожна вершина схеми алгоритму, чи то операційна, чи то умовна, кодується. Причому якщо операційна або умовна вершини повторюються, то кожній з них далі присвоюється її попередній код. У даній схемі алгоритму (рис. 9.2) є три операційні вершини, кожна з яких повторюється двічі, а саме: $i = 1$; $i = i + 1$; $kol = kol + 1$. Кодування вершин подане у табл. 9.1.

Таблиця 9.1 – Таблиця кодування вершин

Код	Зміст	Примітка
mY_1	$i = 1$	
mY_2	$kol = 0$	
mY_3	$A[i]$	Введення $A[i]$
mY_4	$kol = kol + 1$	
mY_5	$i = i + 1$	
mY_6	$B[i]$	Введення $B[i]$
mY_7	kol	Виведення kol
X_1	$A\,[i]\ mod\ 2 = 0$	так – 1, ні – 0
X_2	$i \le n$	так – 1, ні – 0
X_3	$B\,[i]\ mod\ 2 = 0$	так – 1, ні – 0
X_4	$i \le m$	так – 1, ні – 0

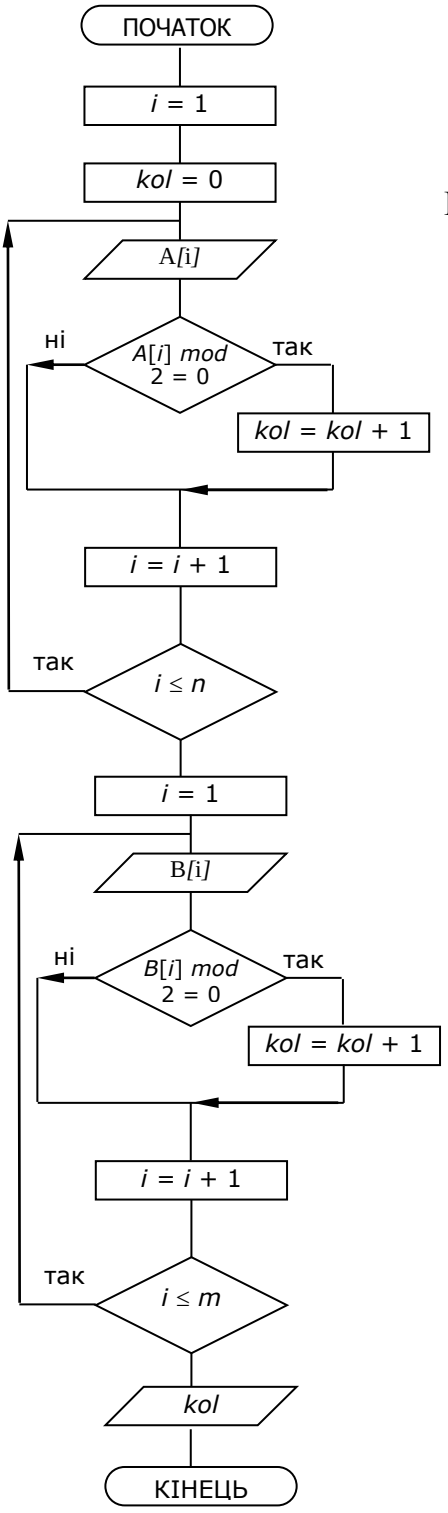


Рисунок 9.2 – Схема алгоритму

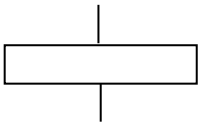


Рисунок 9.3 – Операційна вершина

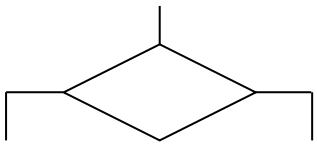


Рисунок 9.4 – Умовна вершина

9.2.3. Побудова закодованої мікрооперaційної схеми алгоритму. Закодована мікрооперaційна схема алгоритму (рис. 9.5.) будується на основі схеми алгоритму (рис. 9.2.) і таблиці кодування операційних та умовних вершин (табл. 9.1) шляхом заміни вмісту блоків схеми алгоритму на відповідні коди мікрооперaцій.

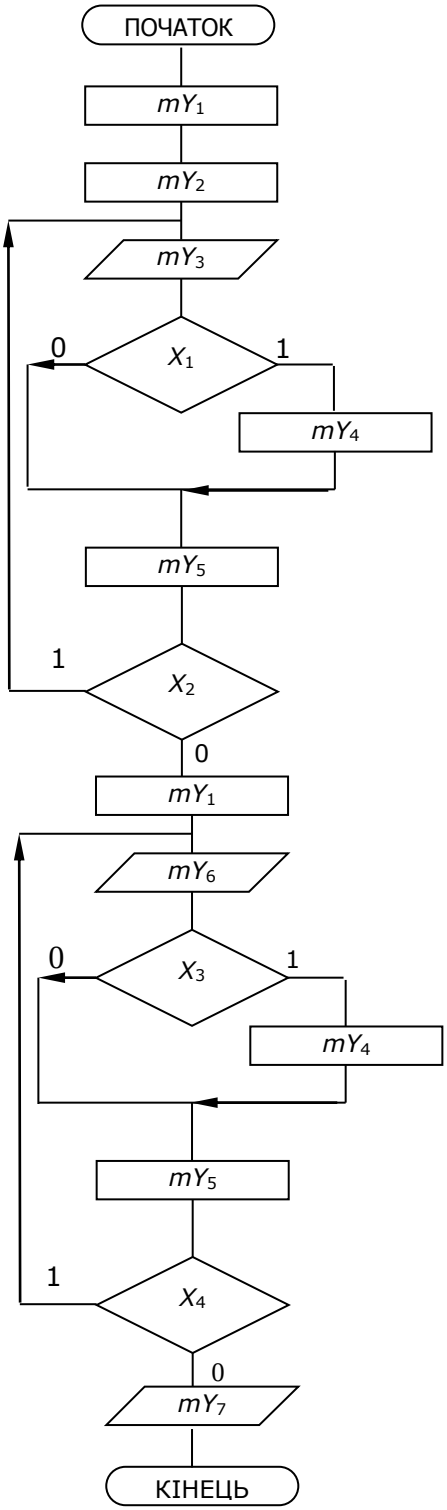


Рисунок 9.5 – Закодована мікрооперaційна схема алгоритму

9.2.4. Побудова таблиці кодування мікрокоманд. Складаємо таблицю кодування мікрокоманд (табл. 9.2). Кожна мікрокоманда кодується своєю мікрооперацією, але мікрооперації, які виконуються одна за одною послідовно протягом одного такту, об'єднуються в одну мікрокоманду. У даному прикладі дві мікрооперації (mY_1 і mY_2) виконуються одна за одною послідовно. Тому вони об'єднані в одну мікрокоманду Y_1.

Таблиця 9.2 – Таблиця кодування мікрокоманд

Мікрокоманда	Мікрооперація
Y_1	mY_1, mY_2
Y_2	mY_3
Y_3	mY_4
Y_4	mY_5
Y_5	mY_1
Y_6	mY_6
Y_7	mY_7

*9.2.5. Побудова закодованої мікрокомандної схеми алгоритму***.** Складаємо закодовану мікрокомандну схему алгоритму, замінюючи мікооперації відповідними мікрокомандами. Проставляємо мітки внутрішніх станів автомата Мілі, а саме так:

- початок та кінець мікрокомандної схеми алгоритму позначаємо міткою s_0;
- мітки s_i проставляємо після кожної мікрокоманди (перед блоком після операційного), де i – порядковий номер мітки.

Якщо на схемі після двох блоків з мікромандами лінії схеми алгоритму сходяться в одну точку, то в такому випадку проставляємо тільки одну мітку внутрішнього стану (рис 9.6).

Закодована мікрокомандна схема алгоритму наведена на рис.9.7.

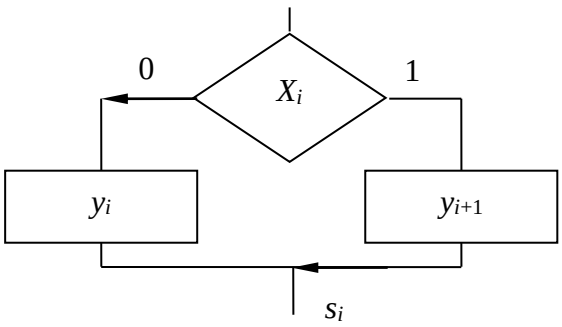


Рисунок 9.6 – Один з варіантів використання міток внутрішніх станів

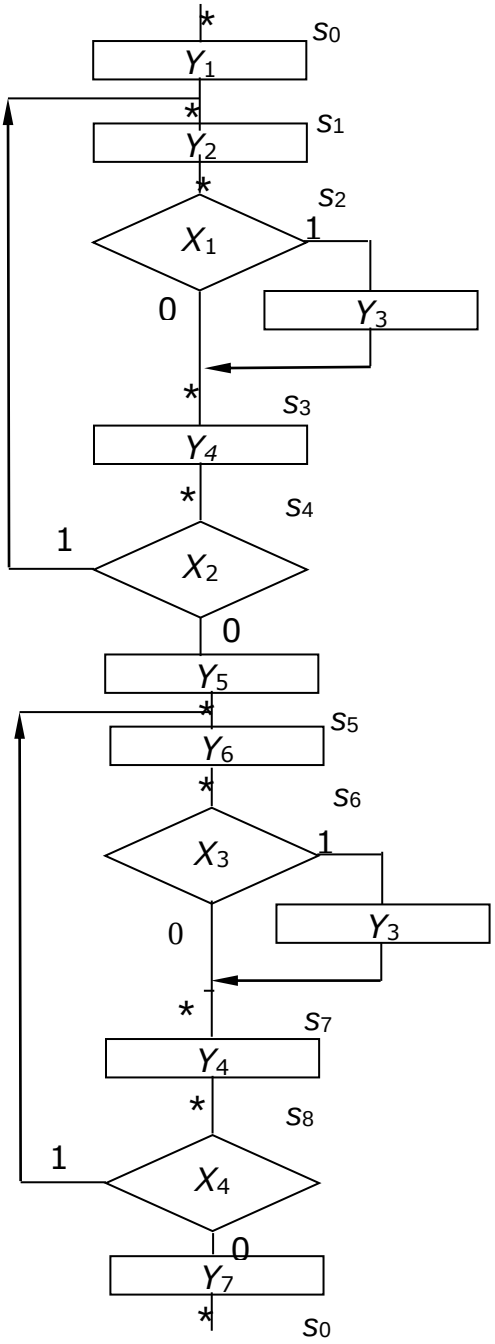


Рисунок 9.7 – Закодована мікрокомандна схема алгоритму

9.2.6. Побудова основної таблиці абстрактного автомата. Будуємо основну таблицю автомата (табл. 9.3). Ця таблиця складається на основі закодованої мікрокомандної схеми алгоритму (рис. 9.7) У першому стовпчику таблиці записуються всі стани, в яких може знаходитись автомат, який розробляється. В першому рядку таблиці в клітинках указуються умови, завдяки яким відбувається перехід автомата з одного стану в інший, а саме вхідні сигнали чи їх комбінація або синхроімпульс (СІ). Перехід за синхроімпульсом відбувається, якщо між двома операційними вершинами не знаходиться умовна вершина. В клітинках таблиці фіксується: стан, до якого переходить автомат, та стан, який з'являється при цьому на виході. Наприклад, із стану s_0 автомат переходить до стану s_1 при подачі на вхід синхроімпульсу. В результаті цього переходу на виході автомата з'явиться сигнал Y_1. Мікрокоманда Y_1 містить мікрооперації mY_1 і mY_2, які і виконуються протягом цього такту. Отже, на перетині рядка s_0 і стовпця СІ = 1 проставляємо – s_1/Y_1.

Таблиця 9.3 – Основна таблиця абстрактного автомата

	СІ=1	X_1	$\overline{X_1}$	X_2	$\overline{X_2}$	X_3	$\overline{X_3}$	X_4	$\overline{X_4}$
s_0	s_1 / Y_1								
s_1	s_2 / Y_2								
s_2		s_3 / Y_3	$s_3 / \_\_$						
s_3	s_4 / Y_4								
s_4				$s_1 / \_\_$	s_5 / Y_5				
s_5	s_6 / Y_6								
s_6						s_7 / Y_3	$s_7 / \_\_$		
s_7	s_8 / Y_4								
s_8								$s_5 / \_\_$	s_0 / Y_7

Якщо при переході із одного стану в інший ніяка мікрокоманда не виконується, то на місці мікрокоманди ставимо тире або Y_0. Так, при переході із стану s_2 в стан s_3 під дією вхідного сигналу $\overline{X_1}$ ніяка мікрокоманда не виконується.

9.2.7. Побудова граф-схеми переходів. Будуємо граф-схему переходів (рис.9.8). Граф-схема переходів будується на основі закодованої мікрокомандної схеми алгоритму (рис. 9.7) і основної таблиці абстрактного автомата (табл. 9.3). Колами позначаються всі внутрішні стани автомата. Стрілки вказують на перехід із стану s_i до стану s_j. Над стрілкою вказується, під дією якого вхідного сигналу або сигналу СІ станеться перехід і який сигнал при цьому з'явиться на виході автомата, тобто яка мікрооперація при цьому має бути виконана.

.

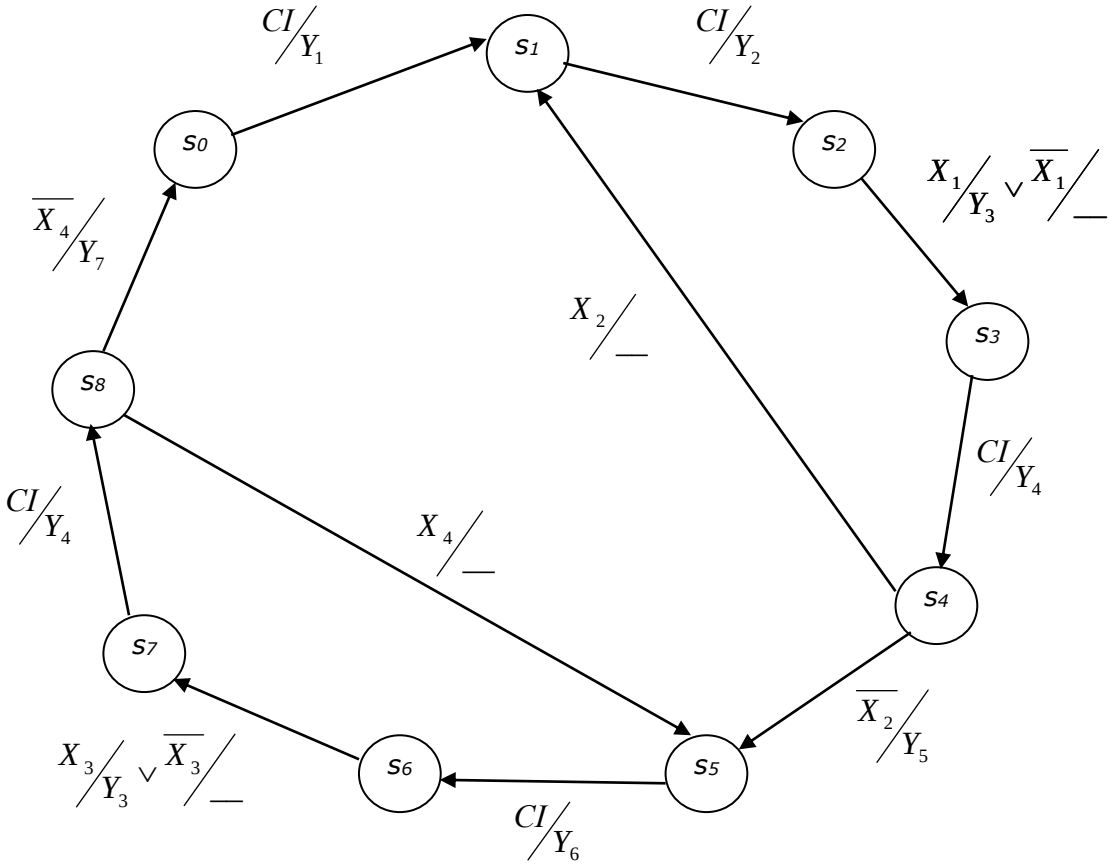


Рисунок 9.8 – Граф-схема переходів

9.2.8. Побудова системи рівнянь функції переходів. Складаємо систему рівнянь функції переходів. Ця система рівнянь складається на основі граф-схеми переходів (рис. 9.8) або на основі основної таблиці абстрактного автомата (табл. 9.3). Сигнал СІ в рівняннях не відображений, тому що його значення дорівнює одиниці.

Наприклад, перехід до стану s_0 зі стану s_8 відбувається при подачі на вхід сигналу $\overline{X_4}$. Тому перше рівняння має такий вигляд: $s_{0t} = s_{8t-1}\overline{x_4}$

Друге рівняння вказує на те, що можливо перейти до стану s_1 зі стану s_0 під дією синхроімпульсу СІ. Крім того, перехід також можливий зі стану s_4, якщо на вхід буде поданий сигнал X_2.

Таким чином, маємо систему рівнянь переходів (9.1):

$$\begin{cases} s_{0t} = s_{8t-1}\overline{X_4} \\ s_{1t} = s_{0t-1} \vee s_{4t-1}X_2 \\ s_{2t} = s_{1t-1} \\ s_{3t} = s_{2t-1}X_1 \vee s_{2t-1}\overline{X_1} = s_{2t-1} \\ s_{4t} = s_{3t-1} \\ s_{5t} = s_{4t-1}\overline{X_2} \vee s_{8t-1}X_4 \\ s_{6t} = s_{5t-1} \\ s_{7t} = s_{6t-1} \\ s_{8t} = s_{7t-1} \end{cases} \quad (9.1)$$

9.2.9. Побудова системи рівнянь функції виходів. Складаємо систему рівнянь для функції виходів. Ця система рівнянь складається на основі закодованої мікрокомандної схеми алгоритму (рис.9.7), де X_i або СІ – вхідний сигнал, а Y_j – вихідний сигнал.

Наприклад, для першого рівняння системи правдивим є твердження, що отримуємо вихідний сигнал Y_{1t} при переході зі стану $s_{0t\text{-}1}$ під впливом синхроімпульсу СІ. Третє рівняння вказує на те, що вихідний сигнал Y_{3t} з'являється при переході зі стану $s_{2t\text{-}1}$ під впливом вхідного сигналу $X1$ або – зі стану $s_{6t\text{-}1}$ під впливом вхідного сигналу X_3. Таким чином, маємо систему рівнянь виходів (9.2):

$$\begin{cases} Y_{1t} = s_{0t-1} \\ Y_{2t} = s_{1t-1} \\ Y_{3t} = s_{2t-1}X_1 \vee s_{6t-1}X_3 \\ Y_{4t} = s_{3t-1} \vee s_{7t-1} \\ Y_{5t} = s_{4t-1}\overline{X_2} \\ Y_{6t} = s_{5t-1} \\ Y_{7t} = s_{8t-1}\overline{X_4} \end{cases} \quad (9.2)$$

9.2.10. Кодування внутрішніх станів автомата. Для того щоб закодувати внутрішні стани автомата, визначаємо кількість необхідних для цього тригерів (n). Кількість тригерів розраховується із співвідношення: $\log_2 S \le n$, де n – кількість тригерів; S – кількість станів s_i .

У нашому прикладі кількість внутрішніх ($s_0 – s_8$) станів $S = 9$.

Тому $n = \log_2 9 \le n = 4$.

Таким чином, необхідно мати 4 тригери. Тому внутрішні стани автомата будемо кодувати чотирирозрядним двійковим кодом. Результат кодування наведено в табл. 9.4, де R_1–R_4 та S_1–S_4 – це входи чотирьох *RS*-тригерів.

Таблиця 9.4 – Кодування внутрішніх станів автомата

	R_1	S_1	R_2	S_2	R_3	S_3	R_4	S_4
s_0	1	0	1	0	1	0	1	0
s_1	1	0	1	0	1	0	0	1
s_2	1	0	1	0	0	1	1	0
s_3	1	0	1	0	0	1	0	1
s_4	1	0	0	1	1	0	1	0
s_5	1	0	0	1	1	0	0	1
s_6	1	0	0	1	0	1	1	0
s_7	1	0	0	1	0	1	0	1
s_8	0	1	1	0	1	0	1	0

9.2.11. Побудова схеми управляючого пристрою (операційного автомата). Операційний автомат (рис. 9.9) є управляючим пристроєм, результатом роботи якого є набір мікрооперацій. Мікрооперації mY_i подаються на вхід виконавчого пристрою, де вони виконуються. Управляючий пристрій (в комп’ютері арифметично-логічний пристрій мікропроцесора) містить набір комбінаційних схем (суматори, регістри, лічильники, компаратори та інше), необхідних для виконання дій, пов’язаних з розв’язанням конкретної задачі. З виконавчого пристрою на вхід операційного автомата подаються сигнали з виходів компараторів X_i. Ці сигнали є вхідними сигналами операційного автомата і формують наступний стан автомата S_i.

Операційний автомат складається з декількох частин.

У *вхідній частині* розташовані чотири *RS*-тригери ($T_1 – T_4$), які зберігають внутрішній стан автомата. Перед *R*-входом кожного тригера розташований логічний елемент ***АБО***. Подача одиничного сигналу «Пуск» на один із входів елементів ***АБО*** встановлює тригери в нульовий стан. Це дозволяє встановити автомат в початковий стан s_0. Також на входи *R* і *S* тригерів подаються сигнали, які є кодами станів автомата в поточний момент часу. Виходи *RS*-тригерів подаються на входи дешифратора.

Дешифратор перетворює двійковий код внутрішнього стану автомата в одиничний сигнал на одному з виходів дешифратора. Виходи дешифратора є станами автомата в попередній момент часу $s_{t\text{-}1}$. Тип дешифратора вибирається залежно від кількості тригерів, які зберігають внутрішній стан автомата. Для даного випадку використовується дешифратор на чотири входи.

Виходи дешифратора $s_{t\text{-}1}$ разом із виходами компараторів X_i подаються на вхід перехідної частини пристрою. На рис. 9.9 використане зображення двох вертикальних сукупних шин. На одну з цих шин подаються сигнали з виходів дешифратора. Наприклад, сигналу $s_{0t\text{-}1}$ припишемо нульовий номер. Оскільки маємо всього дев’ять внутрішніх станів, то використаємо дев’ять виходів дешифратора, позначених номерами від 0 до 8. На другу шину подаються виходи компараторів. Кожний компаратор відповідає одній умовній вершині. Оскільки в даному випадку є чотири компаратори і через те що використовується прямий та інверсний виходи компаратора, то на другу шину подаються вісім сигналів. Номеру 1 відповідає сигнал X_1, номеру 2 відповідає сигнал $\overline{X_1}$ і т.д.

Перехідна частина будується на основі системи рівнянь для функцій переходів. Наприклад, для реалізації рівняння

$$s_{0t} = s_{8t-1}\overline{X_4}$$

потрібний логічний елемент ***І***, на вхід якого необхідно подати восьмий сигнал з першої шини, а саме $s_{8t\text{-}1}$, та восьмий сигнал з другої шини, а саме $\overline{X_4}$. Використавши наступні рівняння, отримаємо перехідну частину автомата. Вихід перехідної частини подається на загальну шину s_t з сигналами, що являються станами автомата в теперішній час.

Вихідна частина будується на основі системи рівнянь (9.2) для функцій виходів. Вона будується подібно перехідній частині, тільки сигнали надходять на вихідну шину Y_t.

У схемі автомата використані дві *програмовані логічні матриці* (ПЛМ).

ПЛМ – конструктивний елемент, виготовлений у вигляді великої інтегральної схеми (ВІС) та призначений для реалізації систем функцій. ПЛМ налагоджується шляхом видалення непотрібних зв'язків за допомогою випалювання або фотошаблона.

На рис. 9.9 верхня ПЛМ використовується для отримання вхідних сигналів для чотирьох *RS*-тригерів, а нижня – для отримання сигналів, що відповідають мікрооперaціям, які має виконувати процесор.

Сигнали, які відповідають станам s_t , подаються на входи верхньої ПЛМ. Виходами цієї ПЛМ є сигнали, які завдяки зворотному зв'язку подаються на входи *RS*-тригерів. Матриця програмується у відповідності до табл. 9.4. Так, якщо маємо одиничне значення стану s_{0t}, необхідно подати одиничний сигнал на входи *R* усіх тригерів. Тому перетин горизонтальної лінії, позначеної номером 0, та вертикальних ліній, позначених номерами 2, 4, 6 і 8 (ці лінії відповідають входам *R* тригерів), позначаємо на схемі крапками. Таким чином, при одиничному сигналі для стану s_{0t} одиничний сигнал надійде на 2, 4, 6 і 8 входи вхідної частини схеми автомата, встановить тригери в нуль і сформує стан $s_{0t\text{-}1}$. Сигнал $s_{0t\text{-}1}$, пройшовши через перехідну частину пристрою, сформує стан s_{1t}, і цей сигнал надійде на вхід з номером 1 ПЛМ, а з виходів 2, 4, 6 і 7 ПЛМ надійде на 2, 4, 6 і 7 входи вхідної частини. Формування інших станів буде проходити аналогічно.

З вихідної шини Y_t сигнали надходять до вихідної ПЛМ. Вона програмується у відповідності до табл. 9.2. Горизонтальні лінії цієї матриці асоціюються з мікрокомандами, а вертикальні – з мікрооперaціями. Так, мікрокоманда Y_{1t} містить дві мікрооперації – mY_1 і mY_2. Тому перетин горизонтальної лінії, позначеної номером 1, та вертикальних ліній, позначених як mY_1 і mY_2, відмічаємо крапками. Таким чином, при формуванні одиничного сигналу для вихідного стану Y_{1t} сигнал надійде на необхідні входи виконавчого пристрою.

9.3. Синтез автомата Мура

Розглянемо синтез автомата Мура на прикладі 2.

Приклад 2. Побудувати операційний автомат, який знаходить максимальний парний елемент у кожному стовпці масиву *A*[*n*,*n*]. Автомат реалізувати за схемою алгоритму у вигляді автомата Мура. Функціональну схему автомата синтезувати на елементах ***І, АБО, НІ*** та *RS*-тригерах, доповнюючи її необхідними за алгоритмом функціональними автоматами.

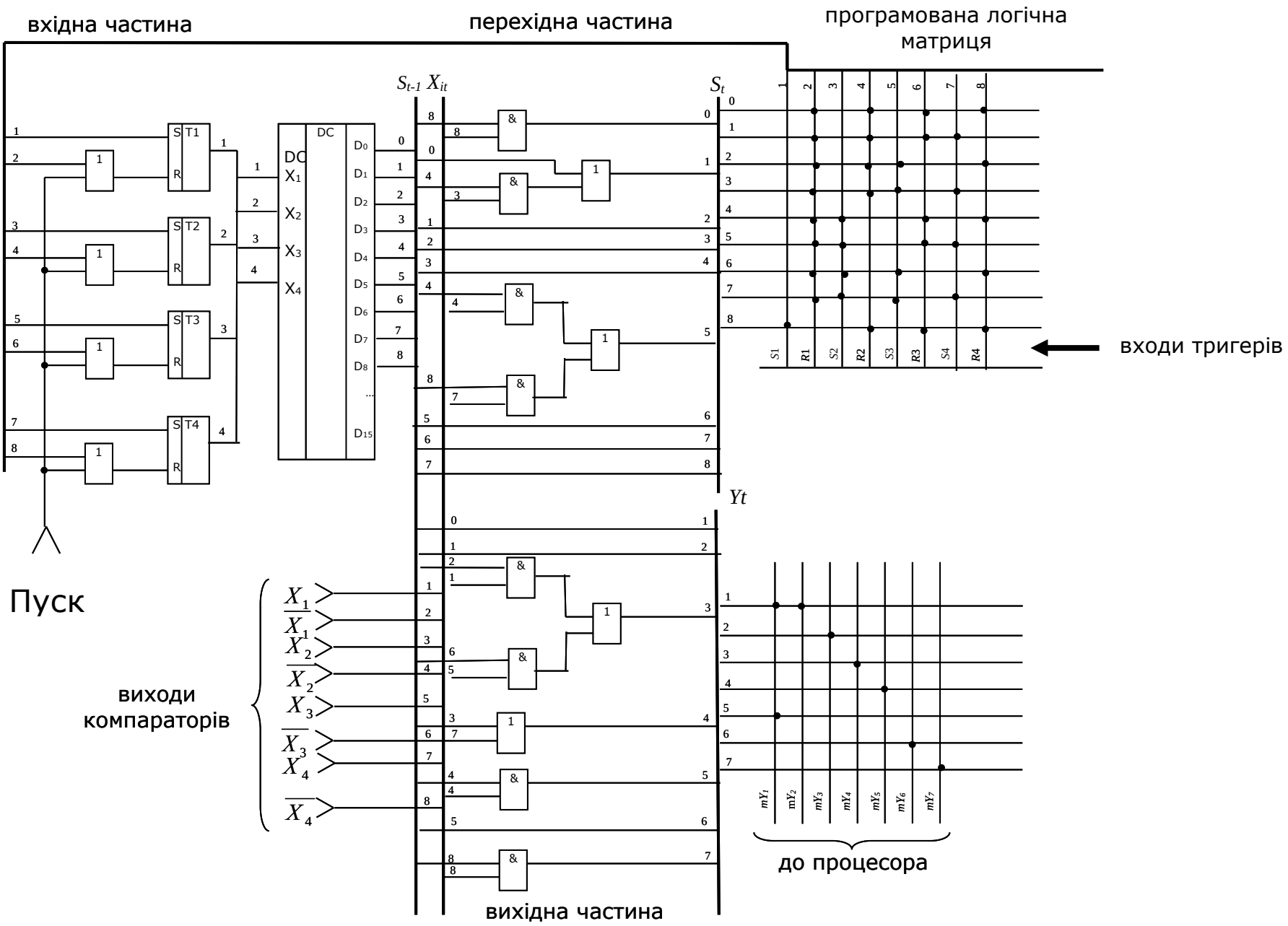


Рисунок 9.9– Схема операціиного автомата

9.3.1. Побудова змістовної схеми алгоритму. Будуємо схему алгоритму (рис. 9.10).

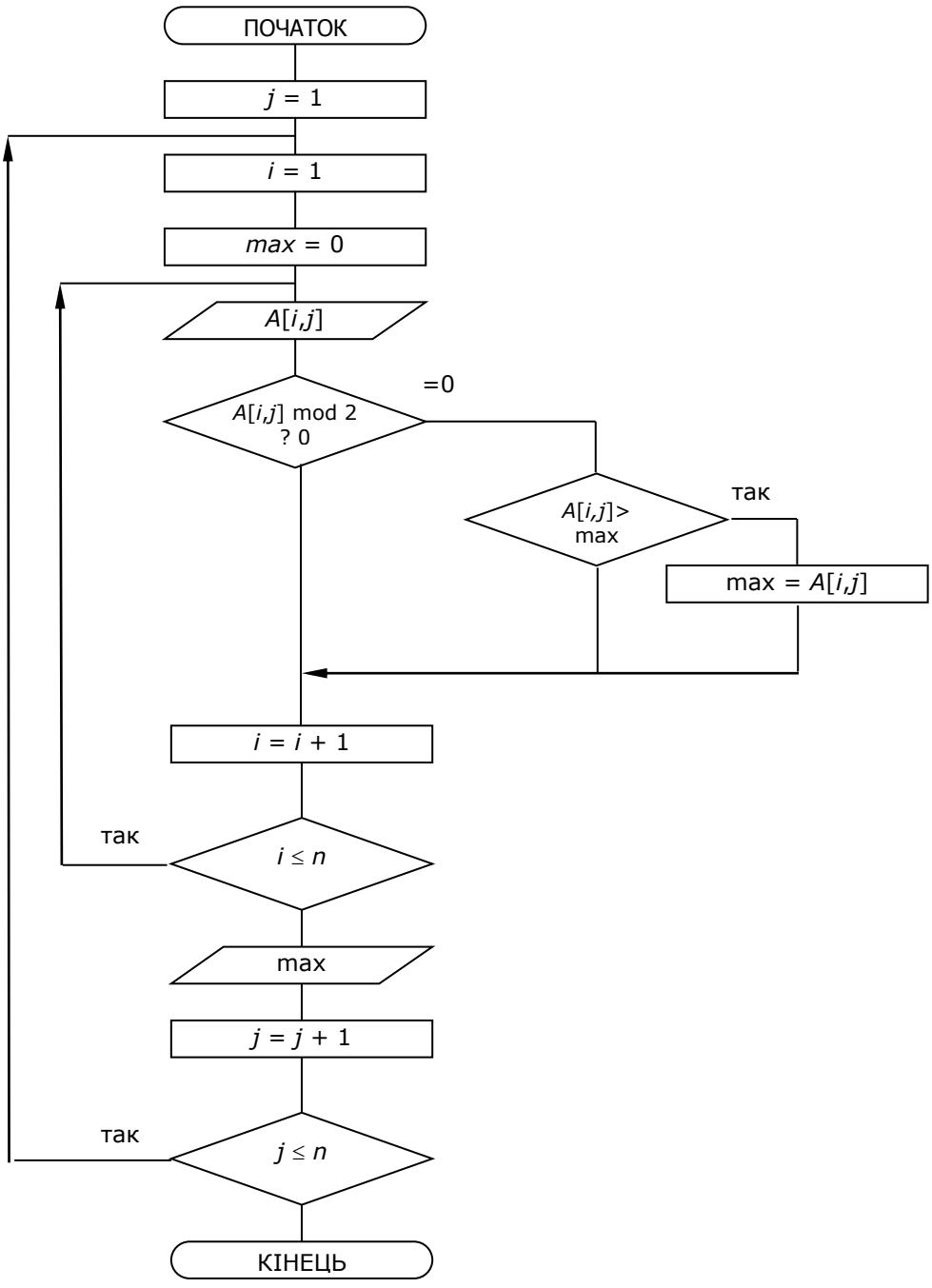


Рисунок 9.10 – Схема алгоритму

9.3.2. Побудова таблиці кодування операційних та умовних вершин

Кожна вершина, чи то операційна, чи то умовна, кодується. Причому якщо операційна або умовна вершини повторюються, то кожній з них далі присвоюється її попередній код. У даній схемі алгоритму (рис. 9.10) вершини, які повторюються, відсутні. Кодування вершин наведене у табл. 9.5.

Таблиця 9.5 – Таблиця кодування вершин

Код	Зміст	Примітка
mY_1	$j = 1$	
mY_2	$i = 1$	
mY_3	max = 0	
mY_4	$A[i,j]$	Введення $A[i,j]$
mY_5	max = $A[i,j]$	
mY_6	$i = i + 1$	
mY_7	max	Виведення max
mY_8	$j = j + 1$	
X_1	$A[i,j] \bmod 2 = 0$	так – 1, ні – 0
X_2	$A[i,j] > \max$	так – 1, ні – 0
X_3	$i \le n$	так – 1, ні – 0
X_4	$j \le m$	так – 1, ні – 0

*9.3.3. Побудова закодованої мікрооперацiйної схеми алгоритму***.** Закодована мікрооперацiйна схема алгоритму (рис. 9.11) будується на основі схеми алгоритму (рис. 9.10) і таблиці кодування операційних та умовних вершин (табл. 9.5) шляхом заміни вмісту блоків схеми алгоритму на відповідні коди мікрооперацій.

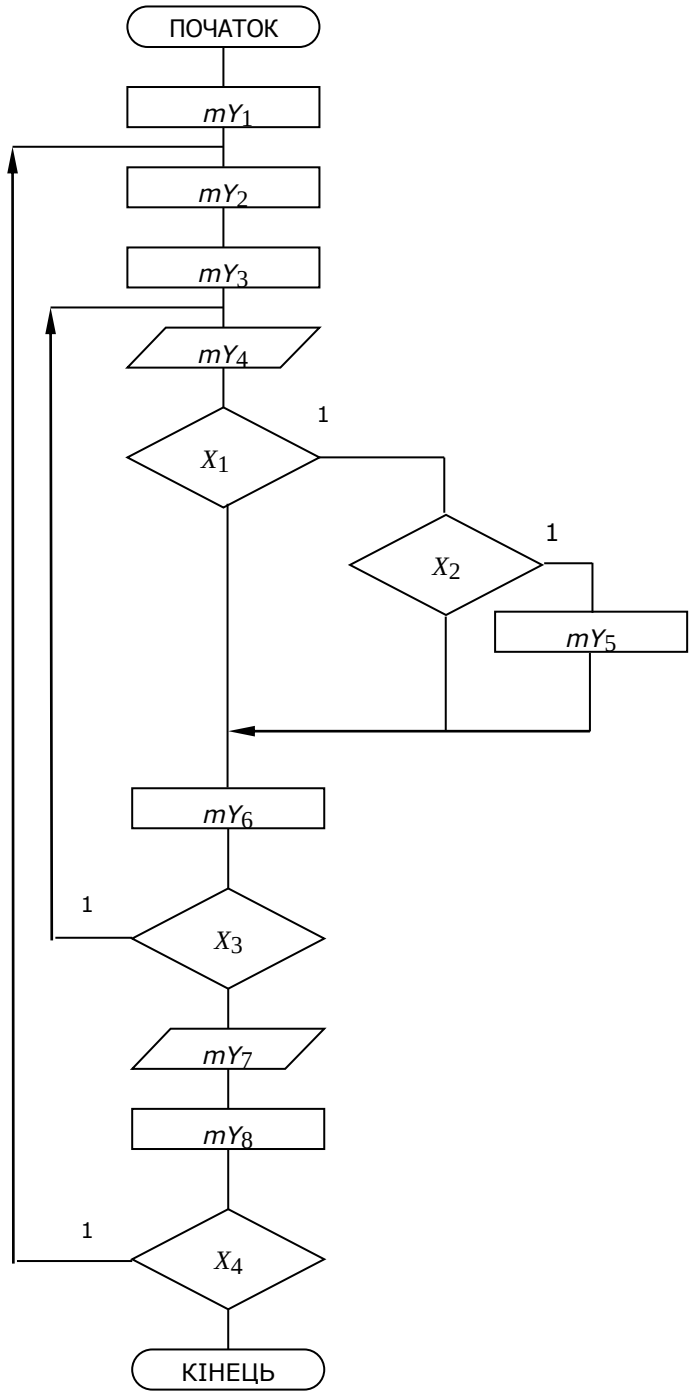


Рисунок 9.11– Закодована мікрооперaційна схема алгоритму

9.3.4. Побудова таблиці кодування операційних та умовних вершин. Складаємо таблицю кодування мікрокоманд (табл. 9.6). Кожна мікрокоманда кодується своєю мікрооперацією, але мікрооперації, які виконуються одна за одною послідовно протягом одного такту, об'єднуються в одну мікрокоманду. У даному прикладі по дві мікрооперації (mY_2 і mY_3) та (mY_7 і mY_8) виконуються одна за одною послідовно. Тому вони об'єднані в одну мікрокоманду, відповідно Y_2 і Y_5.

Таблиця 9.6 – Таблиця кодування мікрокоманд

Мікрокоманда	Мікрооперація
Y_1	mY_1
Y_2	mY_2, mY_3
Y_3	mY_4
Y_4	mY_5
Y_5	mY_6
Y_6	mY_7, mY_8

9.3.5. Побудова закодованої мікрокомандної схеми алгоритму. Закодовану мікрокомандну схему алгоритму наведено на рис. 9.12. Проставляємо мітки внутрішніх станів автомата Мура таким чином:

- мітки ставляться біля кожної мікрокоманди;
- початок і кінець мікрокомандної схеми алгоритма позначається міткою s_0;
- перехід з одного стану в інший повинен проходити через умовну або операційну вершину автомата;
- біля кожної мікрокоманди мітки проставляються відповідно до порядкового номера.

9.3.6. Побудова основної таблиці абстрактного автомата. Основна таблиця абстрактного автомата (табл. 9.7) будується на основі закодованої мікрокомандної схеми алгоритму (рис. 9.12). В першому стовбці таблиці записуються всі стани, в яких може знаходитись автомат. У першому рядку таблиці вказуються вхідні сигнали, їх комбінації або синхроімпульс (СІ). Перехід за синхроімпульсом відбувається, якщо між двома операційними вершинами не знаходиться умовна вершина.

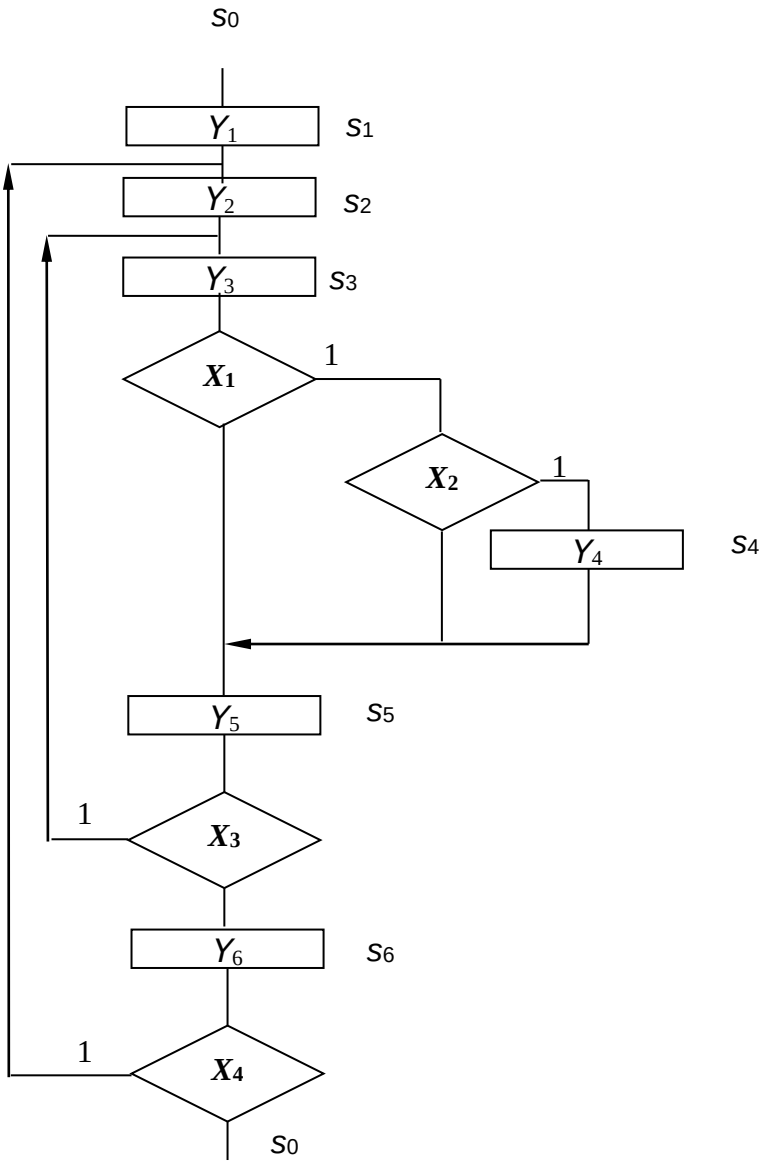


Рисунок 9.12 – Закодована мікрокомандна схема алгоритму

У клітинках таблиці фіксується: стан, до якого переходить автомат та стан, який з'являється при цьому на виході. Наприклад, із стану s_0 автомат може здійснити перехід до стану s_1, і в результаті цього переходу на виході автомата з'явиться вихідний стан Y_1. Таким чином, автомат виконає ті мікрооперації, які виконуються протягом одного такту (мікрооперація mY_1 закодована мікрокомандою Y_1), і цей перехід відбудеться в результаті дії вхідного сигналу СІ.

Таблиця 9.7 – Основна таблиця абстрактного автомата

	CI=1	$\overline{X_1}$	$X_1 X_2$	$X_1 \overline{X_2}$	X_3	$\overline{X_3}$	X_4	$\overline{X_4}$
s_0	s_1 / Y_1							
s_1	s_2 / Y_2							
s_2	s_3 / Y_3							
s_3		s_5 / Y_5	s_4 / Y_4	s_5 / Y_5				
s_4	s_5 / Y_5							
s_5					s_3 / Y_3	s_6 / Y_6		
s_6							s_2 / Y_2	$s_0 / \_\_$

9.3.7. Побудова граф-схеми переходів. Граф-схема переходів (рис. 9.13) будується на основі закодованої мікрокомандної схеми алгоритму (рис. 9.12) і основної таблиці абстрактного автомата (табл. 9.7). Колами позначаються можливі стани автомата. Стрілки вказують на перехід із стану s_i до стану s_j. Над стрілкою вказується, під дією якого вхідного сигналу або сигналу CI станеться перехід і який сигнал при цьому з'явиться на виході автомата.

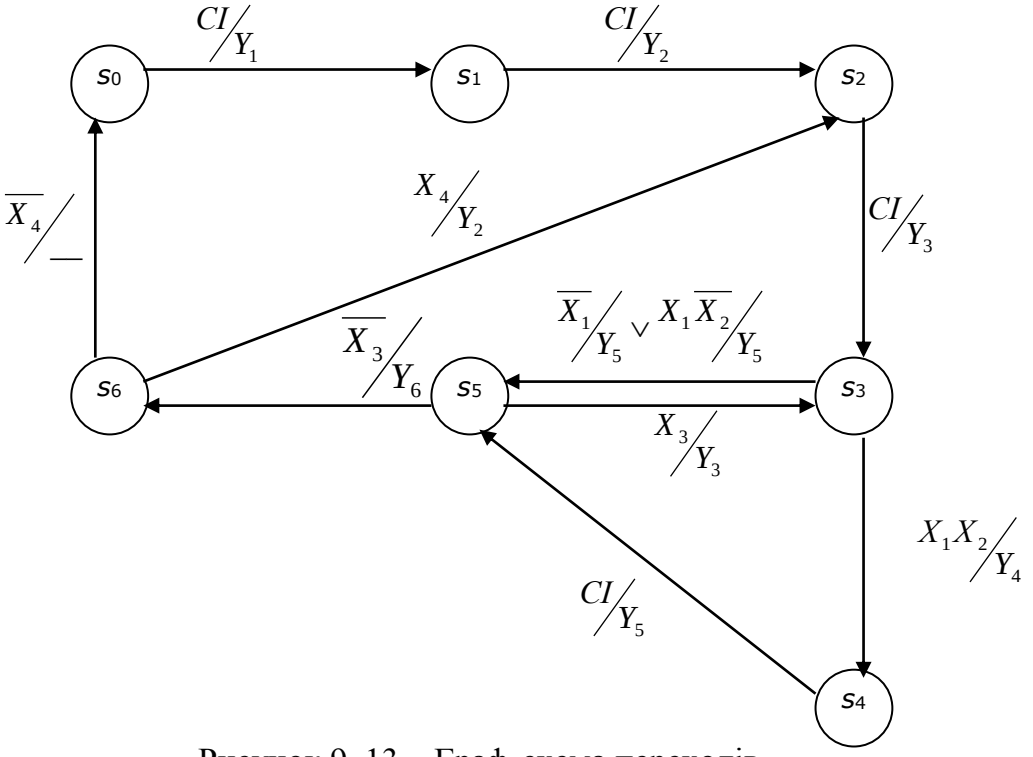


Рисунок 9. 13 – Граф-схема переходів

9.3.8. Побудова системи рівнянь функцій переходів. Складаємо систему рівнянь функцій переходів (9.3). Ця система рівнянь складається на основі граф-схеми переходів (рис. 9.13) або основної таблиці абстрактного автомата (табл. 9.7). Сигнал СІ в рівняннях не відображений. Наприклад, для першого рівняння системи правдивим є твердження, що можна перейти до стану s_0 зі стану s_6, якщо на вхід буде поданий сигнал $\overline{X_4}$. Шосте рівняння вказує на те, що можна перейти до стану s_5 зі стану s_4 під впливом синхроімпульсу або – зі стану s_3, якщо на вхід буде поданий сигнал $\overline{X_1}$ або одночасно сигнали X_1 і $\overline{X_2}$. Таким чином, маємо:

$$\begin{cases} s_{0t} = s_{6t-1}\overline{X_4} \\ s_{1t} = s_{0t-1} \\ s_{2t} = s_{1t-1} \vee s_{6t-1}X_4 \\ s_{3t} = s_{2t-1} \vee s_{5t-1}X_3 \\ s_{4t} = s_{3t-1}(X_1X_2) \\ s_{5t} = s_{4t-1} \vee s_{3t-1}\left(\overline{X_1} \vee X_1\overline{X_2}\right) \\ s_{6t} = s_{5t-1}\overline{X_3} \end{cases} \qquad (9.3)$$

Система рівнянь виходів для автомата Мура не будується.

9.3.9. Кодування внутрішніх станів автомата. Для того щоб закодувати внутрішні стани автомата, визначаємо кількість необхідних для цього тригерів (n). Кількість тригерів розраховується із співвідношення: $\log_2 S \le n$, де n – кількість необхідних тригерів; S – кількість міток s_i (в прикладі $s_0 - s_6$)

$$S = 7, \qquad \log_2 7 \le n \qquad \Rightarrow \qquad n = 3.$$

Таким чином, необхідно мати 3 тригери. Тому внутрішні стани автомата будемо кодувати 3-розрядним двійковим кодом. Результати кодування наведені в табл. 9.8. У таблиці імена R_1–R_4 та S_1–S_4 – це входи трьох RS-тригерів.

Таблиця 9.8 – Кодування внутрішніх станів автомата

	R_1	S_1	R_2	S_2	R_3	S_3
s_0	1	0	1	0	1	0
s_1	1	0	1	0	0	1
s_2	1	0	0	1	1	0
s_3	1	0	0	1	0	1
s_4	0	1	1	0	1	0
s_5	0	1	1	0	0	1
s_6	0	1	0	1	1	0

9.3.10. Побудова схеми операційного автомата. Операційний автомат (рис. 9.14) являється управляючим пристроєм, результатом роботи якого є набір мікрооперацій. Мікрооперації mY_i подаються на вхід виконавчого пристрою, де вони виконуються. Виконавчий пристрій (в комп'ютері арифметично-логічний пристрій мікропроцесора) містить набір комбінаційних схем (суматори, регістри, лічильники, компаратори та інше), необхідних для виконання дій, пов'язаних з розв'язанням конкретної задачі. З виконавчого пристрою на вхід операційного автомата подаються сигнали з виходів компараторів X_i. Ці сигнали є вхідними сигналами операційного автомата і формують наступний стан автомата S_i.

Операційний автомат складається з трьох частин (рис. 9.14).

У *вхідній частині* розташовані три *RS*-тригери (T_1 – T_3), які зберігають внутрішній стан автомата. Перед *R*-входом кожного тригера розташовано логічний елемент ***АБО***. Подача одиничного сигналу на один із входів елементів ***АБО*** встановлює тригери в нульовий стан. Це дозволяє встановити автомат у початковий стан s_0. Також на входи *R* і *S* тригерів подаються сигнали, які є кодами станів автомата в поточний момент часу. Виходи *RS*-тригерів подаються на входи дешифратора. Дешифратор перетворює двійковий код внутрішнього стану автомата в одиничний сигнал на одному з виходів дешифратора. Виходи дешифратора є станами автомата в попередній момент часу s_{t-1}. Тип дешифратора вибирається залежно від кількості тригерів, які зберігають внутрішній стан автомата. Для даного прикладу використовується дешифратор на три входи.

Виходи дешифратора s_{t-1} разом із виходами компараторів X_i подаються на вхід перехідної частини пристрою. На рис. 9.14 використане зображення двох загальних шин. На одну з цих шин подаються сигнали з виходів дешифратора. Наприклад, сигналу s_{0t-1} припишемо нульовий номер. Оскільки маємо всього сім внутрішніх станів, то використаємо сім виходів дешифратора, позначених номерами від 0 до 6. На другу шину подаються виходи компараторів. Кожний компаратор відповідає одній умовній вершині. Оскільки в даному випадку є чотири компаратори з прямими та інверсними виходами, то на другу шину подаються вісім сигналів. Номеру 1 відповідає сигнал X_1, номеру 2 відповідає сигнал $\overline{X_1}$ і т.д.

Перехідна частина будується на основі системи рівнянь для функцій переходів. Наприклад, для реалізації рівняння

$$s_{0t} = s_{6t-1}\overline{X_4}$$

потрібний логічний елемент ***І***, на вхід якого необхідно подати шостий сигнал з першої шини, а саме s_{6t-1}, та восьмий сигнал з другої шини, а саме $\overline{X_4}$. Використавши такі рівняння, отримаємо перехідну частину автомата. Вихід перехідної частини подається на загальну шину s_t з сигналами, що являються станами автомата в теперішній час.

У схемі автомата використані дві *програмовані логічні матриці* (ПЛМ).

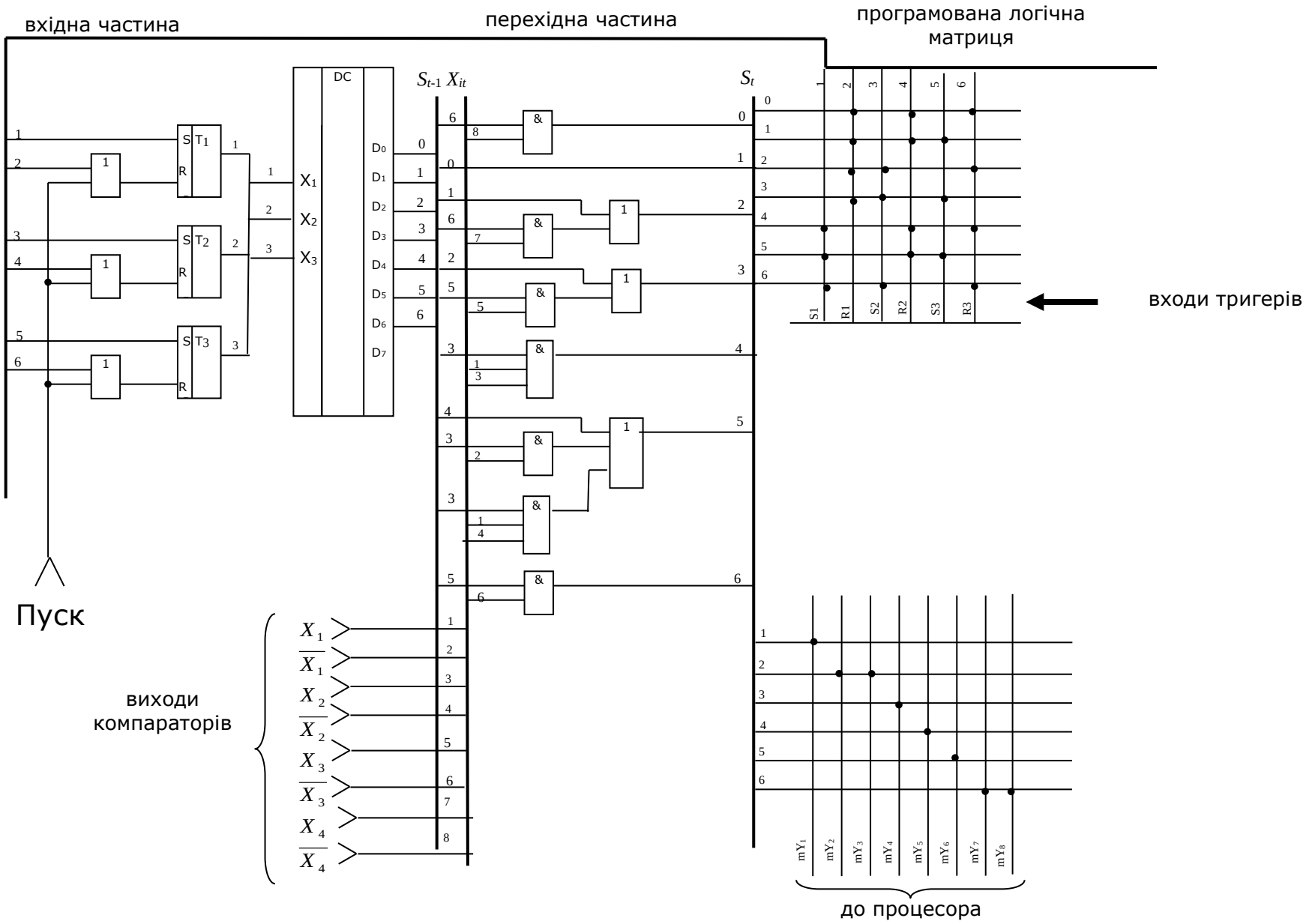


Рисунок 9.14 – Схема операційного автомата

На рис. 9.14 верхня ПЛМ використовується для отримання вхідних сигналів для трьох *RS*-тригерів, а нижня – для отримання сигналів, що відповідають мікрооперaціям, які має виконувати виконавчий пристрій.

Сигнали, які відповідають станам *St*, подаються на входи верхньої ПЛМ. Виходами цієї ПЛМ є сигнали, які завдяки зворотному зв'язку подаються на входи *RS*-тригерів. Матриця програмується у відповідності до табл. 9.8. Так, якщо маємо одиничне значення стану s_{0t}, необхідно подати одиничний сигнал на *R* входи всіх тригерів. Тому перетин горизонтальної лінії, позначеної номером 0, та вертикальних ліній, позначених номерами 2, 4 і 6 (ці лінії відповідають *R* входам тригерів), відмічаємо на схемі крапками. Таким чином, при одиничному сигналі для стану s_{0t}, одиничний сигнал надійде на 2, 4 і 6 входи вхідної частини схеми автомата і встановить тригери в нульовий стан. Сигнал $s_{0t\text{-}1}$, пройшовши через перехідну частину пристрою, сформує стан s_{1t}, і цей сигнал надійде на перший вхід ПЛМ. Далі з 2, 4 і 5 виходів ПЛМ надійде на 2, 4 і 5 входи вхідної частини. Формування інших станів буде проходити аналогічно.

Вихідні стани автомата є станами автомата s_t в теперішній час. Вони подаються також на другу ПЛМ. Вона програмується у відповідності до табл. 9.6. Горизонтальні лінії цієї матриці асоціюються з мікрокомандами, а вертикальні – з мікрооперaціями. Так, мікрокоманда Y_{2t} містить дві мікрооперації – mY_2 і mY_3. Тому перетин горизонтальної лінії, позначеної номером 2, та вертикальних ліній, позначених як mY_2 і mY_3 відмічаємо крапками. Таким чином, при формуванні одиничного сигналу для вихідного стану Y_{2t} сигнал надійде на необхідні входи процесора.

Контрольні запитання

1. Як взаємодіють управляюча і виконавча частини пристрою?
2. Чим відрізняються автомат Мілі та автомат Мура?
3. Яка послідовність дій необхідна для побудови управляючого пристрою для автомата Мілі чи автомата Мура?
4. Як виконується етап кодування операційних та умовних вершин схеми алгоритму кодами мікрооперацій і кодами вхідних сигналів?
5. Як перейти від схеми алгоритму до закодованої мікрооперaційної схеми алгоритму?
6. Як виконується етап кодування мікрокоманд з використанням кодів мікрооперацій?
7. Як отримати закодовану мікрокомандну схему алгоритму?
8. Вміст основної таблиці абстрактного автомата.
9. Як побудувати граф-схему переходів автомата із одного внутрішнього стану в інший?
10. Як побудувати систему рівнянь для функцій переходів і для функцій виходів автомата?
11. Як виконується кодування внутрішніх станів автомата?
12. Реалізація схеми операційного автомата.
13. Опишіть особливості функціонування схеми операційного автомата.

10. ФОРМАЛЬНІ МОВИ І ГРАМАТИКИ

10.1. Визначення формальних мов і граматик

Для побудови компілятора необхідне однозначне та точне задання вхідної та вихідної мов. Таке задання потребує визначення правил побудови допустимих конструкцій мови. Множину таких правил називають **синтаксисом мови**. Крім того, таке задання повинно включати визначення змісту кожної мовної конструкції. Таке визначення називають **семантикою мови.** Для побудови точних і недвозначних описів використовують метод абстракцій. При цьому вхідний тест розглядається як послідовність символів, яка побудована згідно з правилами граматики. Математичні моделі, що використовують подання текстів у вигляді послідовності символів, називають **формальними мовами і граматиками**.

Визначення. Кінцева множина символів, неподільних у даному розгляді, називається *словником* чи *алфавітом*, а символи, що входять у множину, – *буквами алфавіту*.

Наприклад, алфавіт A = {2, *b*, *c*, - , ?} містить 5 букв, а алфавіт B = {00, 01, 10, 11} містить 4 букви, кожна з яких складається з двох символів.

Визначення. Послідовність букв алфавіту називається *словом* чи *ланцюжком* у цьому алфавіті. Число букв, що входять у слово, називається його *довжиною*.

Наприклад, слово $a = bbbc$ в алфавіті A має довжину $l(a) = 4$, а слово b = 0000110010 в алфавіті B має довжину $l(b) = 5$.

Якщо заданий алфавіт A, то позначимо A^* множину всяких ланцюжків, що можуть бути побудовані з букв алфавіту A. При цьому передбачається, що порожній ланцюжок, який позначимо знаком $, також входить у множину A^*.

Визначення. Формальною граматикою Г, що породжує ланцюжок символів, називається така сукупність чотирьох об'єктів:

$$\Gamma = \{ I,\ V_т, V_а, R\},$$

де $V_т$ – термінальний алфавіт (словник); букви цього алфавіту називаються термінальними символами; з них будуються ланцюжки, породжувані граматикою; $V_а$ – нетермінальний, допоміжний алфавіт (словник); букви цього алфавіту використовуються при побудові ланцюжків; вони можуть входити в проміжні ланцюжки, але не повинні входити в результат породження; I – початковий символ граматики, $I \in V_а$; R – множина правил граматики чи правил виведення вигляду $\alpha \to \beta$, де α і β – ланцюжки, побудовані з букв алфавіту $V_т \cup V_а$, що називають повним алфавітом (словником) граматики Г.

До множини правил граматики можуть також входити правила з порожньою правою частиною вигляду $E \to$. Щоб уникнути невизначеності через відсутність символу в правій частині правила, умовимося використовувати символ порожнього ланцюжка, записуючи таке правило у вигляді $E \to \$$.

Щоб встановити закономірності побудови правил граматики, введемо такі поняття.

Визначення. Нехай $\tau \to \gamma$ – правило граматики Г і $\alpha \to \chi'\tau\chi''$ – ланцюжок символів, причому χ', $\chi'' \in (V_т \cup V_а)$. Тоді ланцюжок $\beta \to \chi'\gamma\chi$ може бути отриманий з ланцюжка α шляхом застосування правила $\tau \to \gamma$. У цьому випадку говорять, що ланцюжок β безпосередньо виведений з ланцюжка α і позначають $\alpha \Rightarrow \beta$.

Визначення. Якщо задана сукупність ланцюжків $\Omega = (\varpi_0, \varpi_1, ..., \varpi_n)$, для якої існує послідовність безпосередніх виведень

$$\varpi_0 \Rightarrow \varpi_1,\ \varpi_1 \Rightarrow \varpi_2,\ ...,\ \varpi_{n-1} \Rightarrow \varpi_n\ ,$$

то таку послідовність називають виведенням ϖ_n з ϖ_0 у граматиці Г і позначають:

$$\varpi_0 \Rightarrow^* \varpi_n.$$

Визначення. Множина кінцевих ланцюжків термінального алфавіту $V_т$ граматики Г, виведених з початкового символу I, називається **мовою, породжуваною граматикою** Г, і позначається $L(\Gamma)$.

$$L(\Gamma) = \{\varpi \in V_т \mid I \Rightarrow^* \varpi\}.$$

10.2. Приклади, що ілюструють первинні поняття

Розглянемо кілька прикладів, що ілюструють уведені поняття.

1. Задана граматика $\Gamma_{10.0}$ і потрібно визначити мову, яка породжується цією граматикою:

$$\Gamma_{10.0}:\ V_т = \{a, b, c\},\quad V_а = \{I\},\quad R = \{I \to bac\}.$$

Схема граматики містить одне правило, тому $\Gamma_{10.0}$ породжує мову з одного слова

$$L(\Gamma_{10.0}) = \{bac\}.$$

2. Задана граматика $\Gamma_{10.1}$. Потрібно визначити мову, яка породжується цією граматикою:

$\Gamma_{10.1}$: $V_т = \{a, b, c, d\}$, $V_а = \{I, B, C\}$,

$R = \{I \to aB$

$B \to Cd$

$B \to dc$

$C \to b$

$C \to \$\}$.

Побудуємо всі виведення в цій граматиці:

$I \Rightarrow aB \Rightarrow aCd \Rightarrow abd,$

$I \Rightarrow aB \Rightarrow aCd \Rightarrow ad,$

$I \Rightarrow aB \Rightarrow adc.$

Отже, мова $L(\Gamma_{10.1}) = \{abd, ad, adc\}$.

3. Задана граматика $\Gamma_{10.2}$. Потрібно визначити мову, яка породжується цією граматикою:

$\Gamma_{10.2}$: $V_т = \{0, 1\}$, $V_a = \{I, A\}$,

$R = \{ I \to 0A1$

$0A \to 00A1$

$A \to \$\}$.

Розглянемо кілька виведень за допомогою правил граматики $\Gamma_{10.2}$. Застосовуючи перше і третє правила, одержуємо

$$I \Rightarrow 0A1 \Rightarrow 01.$$

Застосовуючи два рази перше правило і третє, маємо

$$I \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 0011.$$

У загальному випадку, застосовуючи k разів перше правило, одержимо в результаті ланцюжок, що містить k нулів і k одиниць.

Отже, мова, яка породжується граматикою $\Gamma_{10.2}$, містить різні ланцюжки, в яких число нулів дорівнює числу одиниць.

4. Задана граматика $\Gamma_{10.3}$. Потрібно визначити мову, яка породжується цією граматикою.

$\Gamma_{10.3}$: $V_т = \{a, b\}$, $V_а = \{I, A\}$,

$R = \{I \rightarrow aA$

$A \rightarrow aA\}$.

Спроба побудови виведення результату в цій граматиці приводить нас до ланцюжка:

$$I \Rightarrow aA \Rightarrow aaA \Rightarrow aaaA \Rightarrow \ldots,$$

який виявляється нескінченним. Іншими словами, граматика $\Gamma_{10.3}$ породжує порожню мову.

10.3. Порожня мова

Визначення. Якщо мова, породжувана граматикою Г, не містить жодного кінцевого ланцюжка (кінцевого слова), то вона називається **порожньою**.

Твердження. Для того щоб мова $L(\Gamma)$ не була порожньою, у множині R повинне бути хоча б одне правило вигляду $R = \{ \chi \rightarrow \psi \}$, де $\psi \in V_т$ і повинно існувати виведення $I \Rightarrow^* \chi$.

10.4. Типи формальних мов і граматик

У теорії формальних мов виділяють 4 типи граматик, яким відповідають 4 типи мов. Такий розподіл називається «граматична структура Хомського». Ці граматики виділяються шляхом накладення обмежень на правила граматики.

10.4.1. Граматики типу 0. Граматики типу 0, що називаються граматиками загального вигляду, не мають ніяких обмежень на правила породження. Будь-яке правило

$$R = \{\chi \rightarrow \psi\}$$

може бути побудоване з використанням довільних ланцюжків η, $\psi \in (V_т \cup V_а)$. Наприклад, $RLW \rightarrow FWT\,x\,Ab \rightarrow xrtHVD$.

10.4.2. Граматики типу 1. Граматики типу 1, що називаються також контекстно-залежними граматиками, не допускають використання будь-яких правил. Правила виведення в таких граматиках повинні мати вигляд:

$$\chi_1 A \chi_2 \rightarrow \chi_1 \omega \chi_2,$$

де χ_1, χ_2 – ланцюжки, можливо порожні, з множини ($V_т \cup V_a$), символ $A \in V_a$ і ланцюжок $\omega \in (V_т \cup V_a)$. Ланцюжки χ_1 і χ_2 залишаються незмінними при застосуванні правила, тому їх називають контекстом (відповідно лівим і правим), а граматику – контекстно залежною.

Граматики типу 1 значно зручніші на практиці, ніж граматики типу 0, оскільки в лівій частині правила замінюється завжди один нетермінальний символ, який можна зв'язати з деяким синтаксичним поняттям, у той час як у граматиці типу 0 можна заміняти відразу кілька символів, у тому числі й термінальних.

Наприклад, граматика $\Gamma_{10.4}$: $V_т = \{a, b, c, d\}$, $V_a = \{I, A, B\}$,

$$
\begin{aligned}
R = \{\ & I \to aAI && (1)\\
& AI \to AAI && (2)\\
& AA \to ABA && (3)\\
& A \to b && (4)\\
& bBA \to bcdA && (5)\\
& bI \to ba\} && (6)
\end{aligned}
$$

є контекстно залежною, оскільки друге і шосте правила мають непорожній лівий контекст, а третє і п'яте правила містять обидва контексти. Виведення у такій граматиці може мати вигляд:

$$I \Rightarrow aAI \Rightarrow aAAI \Rightarrow abAI \Rightarrow abbI \Rightarrow abba.$$

10.4.3. Граматики типу 2. Граматики типу 2 називають контекстно вільними (КВ) граматиками, або безконтекстними граматиками.

Правила виведення таких граматик мають вигляд:

$$A \to \alpha,$$

де $A \in V_a$ і $\alpha \in (V_т \cup V_a)$.

Очевидно, що ці правила випливають із правил граматики типу 1 за умови, що $\chi_1 = \chi_2 = \$$. Оскільки контекстні умови відсутні, то правила КВ-граматик виходять простіші, ніж правила граматик типу 1. Саме такі граматики використовують для опису мов програмування. Прикладом КВ-граматики може служити така граматика.

$\Gamma_{10.5}$: $V_т = \{a, b\}$, $V_a = \{I\}$,

$$
\begin{aligned}
R = \{ & I \to aIa && (1)\\
& I \to bIb && (2)\\
& I \to aa && (3)\\
& I \to bb\}. && (4)
\end{aligned}
$$

Ця граматика породжує мову, що складається з ланцюжків, кожний з яких у свою чергу складається з двох частин: ланцюжка $\beta \in V_т$ і дзеркального відображення цього ланцюжка β'.

За допомогою правил цієї граматики може бути побудований, наприклад, ланцюжок:

$$I \Rightarrow aIa \Rightarrow abIba \Rightarrow abaIaba \Rightarrow ababbaba.$$

10.4.4. Граматики типу 3. Граматики типу 3 називають автоматними граматиками (А-граматиками). Правила виведення в таких граматиках мають вигляд:

$$A \to a, \text{ або } A \to aB, \text{ або } A \to Ba,$$

де $a \in V_т$, і $A, B \in V_а$, причому граматика може мати тільки правила вигляду $A \to aB$ – правосторонні правила, або тільки вигляду $A \to Ba$ – лівосторонні правила. Прикладами автоматних граматик можуть бути правостороння граматика $Г_{10.6}$ і лівостороння граматика $Г_{10.7}$.

$Г_{10.6}$: $V_т = \{a, b\}$, $V_а = \{I, A, Z\}$,

$$\begin{aligned} R = \{ & I \to aI && (1)\\ & I \to aA && (2)\\ & A \to bA && (3)\\ & A \to aZ && (4)\\ & A \to bZ && (5)\\ & Z \to \$\}. && (6) \end{aligned}$$

$Г_{10.7}$: $V_т = \{a, b\}$, $V_а = \{I, A, Z\}$,

$$\begin{aligned} R = \{ & I \to Ab && (1)\\ & A \to Ab && (2)\\ & A \to Za && (3)\\ & Z \to Za && (4)\\ & Z \to \$\}. && (5) \end{aligned}$$

Ці граматики є еквівалентними і породжують мову

$$L(Г_7) = \{a...ab...b \mid n, m \ \ 0\}.$$

Між множинами мов різних типів існує відношення включення:

$$\{L_{\text{типу 3}}\} \subset \{L_{\text{типу 2}}\} \subset \{L_{\text{типу 1}}\} \subset \{L_{\text{типу 0}}\}.$$

Доведено, що існують: мови типу 0, які не є мовами типу 1; мови типу 2, які не є мовами типу 1; мови типу 3, які не є мовами типу 2.

З огляду на те, що найбільш практично застосовуються граматики типу 2 і типу 3, подальший виклад присвячується розгляду саме цих типів граматик.

10.5. Виведення у КВ-граматиках і правила побудови дерева виведення

Формальні граматики дозволяють задавати мови, що являють собою множину ланцюжків, побудованих за визначеними правилами. Використовуваний спосіб завдання дозволяє будь-як побудувати ланцюжок, що належить мові. Щоб зробити процес побудови, що називається наочним

виведенням, його зображають у вигляді графа, точніше, у вигляді дерева, яке називають синтаксичним деревом, або деревом виведення. З огляду на те, що виведення будь-якого ланцюжка, який належить мові, породжуваній заданою граматикою, повинно починатися з початкового символу, правила побудови дерева можна сформулювати так:

1) як початок чи вершину кореня дерева візьмемо вершину, яку позначимо початковим символом граматики *I*; ця вершина утворить нульовий ярус дерева;

2) якщо при виведенні ланцюжка на черговому кроці використовується правило граматики $A \to \alpha$ і вершина, позначена нетермінальним символом *A*, розташована на ярусі з номером k – 1, то до побудованого дерева потрібно додати стільки вершин, скільки міститься символів у ланцюжку α; треба розташувати ці вершини на ярусі k , позначити їх символами ланцюжка α і з'єднати ці вершини дугами з вершиною *A*.

Результатом виведення по дереву є множина кінцевих вузлів – листів, що виписуються при обході дерева ліворуч – униз і праворуч – нагору.

Розглянемо, наприклад, граматику $\Gamma_{10.10.}$

$\Gamma_{10.8}$: $V_т = \{a, b\}$, $V_а = \{I\}$,

$R = \{I \to aIb$ (1)

$I \to ab\}$, (2)

яка породжує мову $L(\Gamma_8) = \{aa...abb...b\}$, де *a* і *b* повторюються по *n* разів (n = 1, 2, ...).

Виведення ланцюжка за допомогою правил цієї граматики має вигляд дерева, відображеного на рис. 10.1.

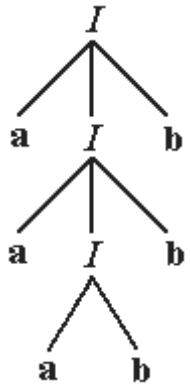


Рисунок 10.1 – Виведення ланцюжка *aaabbb*

10.5.1. Синтаксичний розбір. Виведення ланцюжка за допомогою правил граматики може бути задане не тільки у вигляді синтаксичного дерева. Якщо пронумерувати правила граматики, то послідовність номерів правил, які використані, також задає виведення.

Визначення. Послідовність номерів правил граматики Г, застосування яких дозволяє побудувати виведення заданого ланцюжка σ з початкового символу граматики, називається синтаксичним розбором ланцюжка σ.

Наприклад, у граматиці $\Gamma_{10.9}$

$\Gamma_{10.9}$: $V_т = \{i, +, *, (,)\}$, $V_а = \{E, T, P\}$

$$R = \{E \to E + T \quad (1)$$
$$E \to T \quad (2)$$
$$T \to T * P \quad (3)$$
$$T \to P \quad (4)$$
$$P \to (E) \quad (5)$$
$$P \to i\}, \quad (6)$$

правила якої пронумеровані, виведення ланцюжка $i * i + i$ буде таким:

$$E \Rightarrow E + T \Rightarrow T + T \Rightarrow T * P + T \Rightarrow P * P + T \Rightarrow i * P + T \Rightarrow i * i + T \Rightarrow$$
$$i * i + P \Rightarrow i * i + i.$$

Отже, послідовність використання правил є такою: 1, 2, 3, 4, 6, 6, 4, 6.

Якщо в процесі побудови виведення з'являються проміжні ланцюжки, що містять кілька нетермінальних символів, то можна продовжувати виведення, заміняючи кожний з ланцюжків. Таким чином, ті самі правила можуть бути використані при виведенні ланцюжка у будь-якому порядку.

Наприклад, виведення ланцюжка $i + i$ в граматиці $\Gamma_{10.9}$ може бути отримане декількома різними способами.

10.5.2. Ліве і праве виведення. Серед різних типів виведення найбільший інтерес становлять наступні два типи виведення.

Визначення. Якщо під час побудови виведення ланцюжка α при застосуванні кожного правила заміняється самий лівий нетермінальний символ, то таке виведення називається лівим, або лівостороннім виведенням α. Але, якщо при побудові виведення α завжди заміняється самий правий нетермінальний символ проміжного ланцюжка, то таке виведення називається правим, або правостороннім виведенням α.

Наприклад, наведене вище виведення ланцюжка $i * i + i$ в граматиці $\Gamma_{10.9}$ є лівостороннім виведенням. Слід зазначити, що різним виведенням ланцюжка $i + i$ в граматиці $\Gamma_{10.9}$ відповідає те ж саме синтаксичне дерево. Аналогічна ситуація має місце і при виведенні ланцюжка $i * i + i$.

10.6. Неоднозначні та еквівалентні граматики

Існують граматики, в яких той самий ланцюжок може бути отриманий за допомогою різних виведень. Наприклад, у граматиці $\Gamma_{10.10}$ ланцюжок abc може бути отриманий за допомогою двох різних виведень, і, отже, їм відповідають два різних синтаксичних дерева.

$\Gamma_{10.10}$: $V_т = \{a, b, c, d\}$, $V_а = \{I, A, B\}$,

$$R = \{I \to AB \quad (1)$$

$A \to a$ (2)

$A \to ac$ (3)

$B \to b$ (4)

$B \to cb\}$. (5)

Перше виведення цього ланцюжка має вигляд:

1) $I \Rightarrow AB \Rightarrow Ab \Rightarrow acb$,

а друге можна одержати так:

2) $I \Rightarrow AB \Rightarrow Acb \Rightarrow acb$.

Цим виведенням відповідають різні синтаксичні дерева і розбори (рис. 6.2).

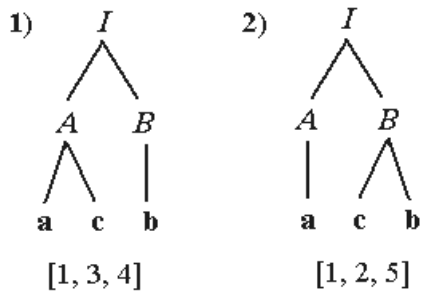


Рисунок 10.2 – Синтаксичні дерева і розбори виразу *acb*

Наступна граматика також допускає побудову одного і того ж ланцюжка за допомогою двох виведень, що мають різні синтаксичні дерева.

$\Gamma_{10.11}$: $V_т = \{0, +\}$, $V_а = \{I\}$,

$R = \{I \to 0$ (1)

$I \to I + 0$ (2)

$I \to 0 + I\}$. (3)

Два виведення цієї граматики, що породжують однакові ланцюжки, мають вигляд:

1) $I \Rightarrow I + 0 \Rightarrow I + 0 + 0 \Rightarrow 0 + 0 + 0$,

2) $I \Rightarrow 0 + I \Rightarrow 0 + 0 + I \Rightarrow 0 + 0 + 0$,

а синтаксичні дерева, що відповідають цим виведенням, зображені на рис. 10.3.

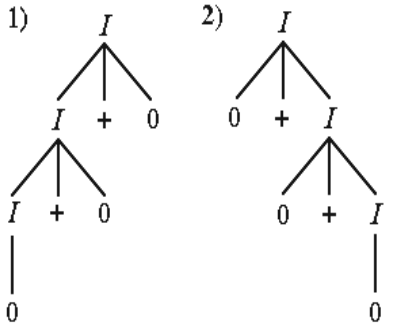


Рисунок. 10.3 – Синтаксичні дерева виведення виразу: 0 + 0 + 0

Розглянута властивість граматик називається *неоднозначністю*. Вона може бути визначена в такий спосіб.

Визначення. Ланцюжок мови $L(\Gamma)$ називається неоднозначним, якщо для його виведення існує більш ніж одне синтаксичне дерево. Якщо граматика Г породжує неоднозначний ланцюжок, то вона називається неоднозначною.

Неоднозначність може існувати не тільки в штучних мовах. Добре відомо, що в природних мовах можуть бути пропозиції, які допускають неоднозначне написання. Наприклад, у фразі "Вино забруднило вікно" не ясно, що є підметом, а що доповненням. Іншим прикладом служить англійська фраза: *"They are flying planes"*, яка може бути зрозуміла подвійно: “Вони пілотують літак” або “Це літаки, що летять”.

Властивість неоднозначності є вкрай небажаною для штучних мов, оскільки вона не дозволяє однозначно побудувати дерево виведення за заданим ланцюжком мови.

У загальному випадку можна зробити такий висновок:

1) кожному ланцюжку, виведеному в граматиці, можуть відповідати одне або кілька синтаксичних дерев;

2) кожному синтаксичному дереву можуть відповідати кілька виведень;

3) кожному синтаксичному дереву відповідають єдине праве і єдине ліве виведення.

Крім того, варто підкреслити, що та сама мова може бути отримана за допомогою різних граматик.

Визначення. Дві граматики – Γ_1 і Γ_2 – називаються еквівалентними, якщо вони породжують одну і ту ж саму мову, тобто $L(\Gamma_1) = L(\Gamma_2)$.

10.7. Способи задання схем граматик

Схема граматики містить правила виведення, які визначають синтаксис мови, або, іншими словами, можливі компоненти і конструкції ланцюжків породжуваної мови. Для задання правил використовуються різні форми опису: символічна, форма Бекуса-Наура, ітераційна форма і синтаксичні діаграми. Але найбільш поширеною є символічна форма задання правил. Вона передбачає використання окремих символів як елементів нетермінального словника і стрілки – як роздільника правої і лівої частин правила.

Задача побудови формальних граматик полягає в тому, що для заданої у вигляді опису природною мовою множини кінцевих ланцюжків, потрібно побудувати формальну граматику, що породжує цю множину ланцюжків. З огляду на те, що термінальний словник граматики повинен включати всі символи, які використовуються для побудови ланцюжків, що входять у задану множину, результатом розв’язання задачі повинні стати нетермінальний словник і схема граматики.

Побудова цих об'єктів є дуже складною, оскільки вона повинна виконуватися неформально і вимагає уявного охоплення всіляких варіантів побудови ланцюжків заданої множини і синтезу правил їх побудови. Побудова ускладнюється ще і тим, що вона, як і всяка інша задача синтезу, має багато розв’язань.

10.7.1. Рекомендації щодо побудови граматик. Основою створення правил граматики є спосіб виділення структури заданої множини ланцюжків. Цей спосіб передбачає розчленування ланцюжків, що входять у задану множину, на їх частини таким чином, щоб виявити повторювані частини ланцюжків і частини, що входять в усі ланцюжки в незмінному вигляді. Таке розчленування на частини являє собою виявлення структури ланцюжків заданої множини.

Для кожного виявленого елемента структури введемо позначення. Множина таких позначень становить основу словника нетермінальних символів деякої граматики. Наступним кроком побудови є виявлення послідовностей, у яких елементи структури можуть входити в задані ланцюжки. Такі послідовності є основою для побудови правил граматики. Щоб показати, яким чином структура ланцюжків відображається в правила граматики, розглянемо такі приклади.

1. Ланцюжку, що складається з заданих символів *abc*, відповідає правило

$$I \to abc.$$

2. Ланцюжку, що починається з заданого символу *a*, відповідає правило

$$I \to aA.$$

3. Ланцюжку, що закінчується заданим символом *a*, відповідає правило

$$I \to Aa.$$

4. Ланцюжку, що починається і закінчується заданими символами *a* і *b*, відповідає правило

$$I \to aAb.$$

5. Ланцюжку, що містить усередині символ *a*, відповідає правило

$$I \to AaB.$$

6. Ланцюжку заданої довжини $l = 2$ відповідають правила

$$A \to aB \text{ і } B \to a.$$

7. Ланцюжку, що складається з повторюваних символів *a*, відповідають правила

$$A \to aA \text{ і } A \to a.$$

10. Ланцюжку, що складається із символів *a* і *b*, які чергуються, відповідають правила

$$A \to aB \text{ і } B \to bA.$$

10.7.2. Опис списків. Розглянемо побудову граматик для послідовностей символів і послідовностей символів з роздільниками, тобто для списків.

1. Позначимо елемент послідовності через *a*. Найпростіша послідовність може складатися з одного елемента *a*. Всі інші послідовності можуть бути отримані шляхом приписування до вже побудованої послідовності ще одного елемента. Якщо позначити побудовану частину послідовності нетермінальним символом *R*, а послідовність – символом *L*, то одержимо правила граматики у вигляді:

$\Gamma_{10.13}$:

$$L \to aR \quad (1)$$

$$R \to aR \quad (2)$$

$$R \to \$. \quad (3)$$

2. У попередній задачі передбачалося, що список L повинен містити хоча б один елемент. Якщо ж припустити, що множина ланцюжків, породжених правилами граматики, може включати порожній символ, то до побудованих правил потрібно додати ще одне правило $L \to \$$. У цьому випадку набір правил має вигляд:

$$\Gamma_{10.14}: \quad L \to aR \qquad (1)$$
$$R \to aR \qquad (2)$$
$$R \to \$ \qquad (3)$$
$$L \to \$. \qquad (4)$$

3. Розглянемо побудову списку, між елементами якого повинні стояти роздільники. Виберемо як роздільник кому. Найпростіший список, як і в попередньому випадку, складається з одного елемента, а побудова списку з декількох елементів може бути виконана приписуванням до вже побудованої частини списку роздільника з елементом списку. Правила, що відповідають цій побудові, мають вигляд:

$$\Gamma_{10.15}: \ L \to aR \qquad (1)$$
$$R \to ,aR \qquad (2)$$
$$R \to \$. \qquad (3)$$

4. Якщо список з роздільниками може бути порожнім, то наведений вище набір правил потрібно доповнити ще одним правилом з порожньою правою частиною. В результаті отримаємо:

$$\Gamma_{10.16}: \quad L \to aR \qquad (1)$$
$$R \to ,aR \qquad (2)$$
$$R \to \$ \qquad (3)$$
$$L \to \$. \qquad (4)$$

У загальному випадку, якщо описана множина ланцюжків, що являють собою деяку мову, і потрібно побудувати граматику, яка породжує цю множину ланцюжків, то слід робити так:

1) виписати кілька прикладів із заданої множини ланцюжків;

2) проаналізувати структуру ланцюжків, виділяючи початок, кінець, що повторюються, або символи з групи символів;

3) ввести позначення для складних структур, що складаються з груп символів; такі позначення є нетермінальними символами граматики, яка шукається;

4) побудувати правила для кожної з виділених структур, використовуючи для задання повторюваних структур рекурсивні правила;

5) об’єднати всі правила;

6) перевірити за допомогою виведень можливість одержання ланцюжків з різною структурою.

10.7.3. Приклад побудови граматики. Застосування наведених рекомендацій розглянемо на такому прикладі. Потрібно побудувати граматику для мови L, термінальний словник якої $V_т = \{*, |\}$, а ланцюжки, що утворюють мову, мають наступну структуру:

а) кожен ланцюжок починається символом * і закінчується двома символами **.

б) між початком і кінцем ланцюжків можуть бути або непорожня послідовність паличок, або кілька таких послідовностей, розділених символами *.

1. Спочатку побудуємо кілька ланцюжків заданої мови, що можуть бути подані в такому вигляді:

* |||**,

* |*|*|**,

* ||*||||*|||||**,

* |||*|*||*||||||**.

2. Розгляд наведених ланцюжків, дає можливість виділити їх наступні структурні компоненти:

- початок ланцюжка (символ *);
- кінець ланцюжка (символи **);
- непорожня група паличок;
- послідовність груп паличок, розділених зірочками.

3. Позначимо групу паличок символом *A*, а послідовність груп паличок символом *B*.

4. Виділені структури можна розглядати як списки. Так, послідовність паличок являє собою список без роздільників, елементом якого є паличка. Правила граматики, що задають такий список, мають такий вигляд:

$$A \to |B$$

$$B \to |B$$

$$B \to \$.$$

Послідовність груп паличок, розділених зірочкою, являє собою список з роздільником. Елементом такого списку є група паличок *A*, а роздільником – зірочка. Правила граматики, які задають такий список, можна записати так:

$$C \to AE$$

$$E \to *AE$$

$$T \to \$.$$

З огляду на те, що кожен ланцюжок мови повинен мати початок і кінець, і, вибираючи як початковий символ граматики *I*, одержуємо правило, що визначає загальний вигляд ланцюжка:

$$I \to *C**.$$

5. Поєднуючи побудовані правила, остаточно одержимо схему граматики у вигляді:

$$\Gamma_{10.17}: \quad R = \{I \to *C**$$

$$C \to AE$$

$$E \to *AE$$

$$E \to \$$$

$$A \to |B$$

$$B \to |B$$

$$B \to \$\}$$

6. За допомогою правил побудованої граматики можна отримати, наприклад, ланцюжок

$$I \Rightarrow *C** \Rightarrow *AE** \Rightarrow *A*AE** \Rightarrow$$

$$*A*A*AE \Rightarrow *A*A*A** \Rightarrow$$

$$*A*A* \mid B** \Rightarrow *A*A* \mid ** \Rightarrow$$

$$*A* \mid B* \mid ** \Rightarrow *A* \mid * \mid ** \Rightarrow$$

$$* \mid B* \mid * \mid ** \Rightarrow * \mid * \mid * \mid **.$$

Побудоване виведення ілюструє можливість породження ланцюжків заданої мови за допомогою побудованої граматики.

Однією з основних областей застосування формальних граматик є опис мов програмування. Розглянемо побудову граматик для основних конструкцій мов програмування. Щоб скоротити розміри граматик, на розглянуті конструкції накладаються деякі, іноді істотні, обмеження.

10.7.4. Граматики, що описують цілі числа без знака та ідентифікатори. Цілі числа являють собою послідовність цифр, тому їх можна розглядати як списки, елементами яких є цифри. Позначимо множину цифр нетермінальним символом *D*. Використовуючи як аналог граматику, що задає список без роздільників, одержимо схему граматики для цілих чисел у вигляді:

$\Gamma_{10.18}$: $R = \{$ $N \to DK$ (1)

$K \to DK$ (2)

$K \to \$$ (3)

$D \to \{0 \mid 1 \mid ... \mid 9\}\ \}$ (4)

Якщо пронумерувати правила, то виведення числа 145 може бути таким: $N \Rightarrow DK \Rightarrow 1K \Rightarrow 1DK \Rightarrow 14K \Rightarrow 14DK \Rightarrow 145K \Rightarrow 145$, тобто матимемо синтаксичний розбір [1, 4.2, 2, 4.5, 2, 4.6, 3]. У зв'язку з тим, що правило 4 об'єднує в собі 10 правил, то позначення 4.2 говорить про використання в даному випадку другого підправила, а саме $D \to 1$.

Структуру ідентифікатора можна подати у вигляді двох компонентів – початку й основної частини. Початком може бути кожна з букв (множину букв позначимо нетермінальним символом *C*), а основна частина (нетермінальний символ *A*) являє собою список без роздільників, елементами якого можуть бути або букви, або цифри (множину цифр позначимо нетермінальним символом *D*). Використовуючи виділені компоненти, одержимо схему граматики такого вигляду:

$\Gamma_{10.19}$: $R = \{I \to CA$ (1)

$A \to CA \mid DA$ (2)

$A \to \$$ (3)

$$D \to 0 \mid 1 \mid ... \mid 9 \qquad (4)$$
$$C \to a \mid b \mid c \mid ... \mid z\}. \qquad (5)$$

Синтаксичний розбір ідентифікатора *ab*12 буде таким: [1, 2.1, 5.1, 2.1, 5.2, 2.2, 4.2, 2.2, 4.3, 3].

10.7.5. Граматики для арифметичних виразів. Умовимося розглядати арифметичні вирази, які використовують тільки ідентифікатор *i* та знаки додавання і множення. Спочатку побудуємо граматику для виразів без дужок. Такі вирази являють собою ланцюжки, які можна розглядати як списки з роздільниками, в яких роль роздільників виконують знаки операцій (нетермінальний символ *W*). Відповідно до цієї аналогії одержуємо схему граматики.

$$\Gamma_{10.20}: R = \{H \to iK \qquad (1)$$
$$K \to WiK \qquad (2)$$
$$K \to \$ \qquad (3)$$
$$W \to + \qquad (4)$$
$$W \to *\}. \qquad (5)$$

Щоб побудувати граматику, яка допускає використання в арифметичних виразах дужок без вкладеності, подамо структуру таких виразів у вигляді списку з роздільниками, елементами якого є вираз (нетермінальний симол *H*) без дужок або вираз, узятий в дужки. Роздільниками цього списку є знаки операцій (нетермінальний символ *W*, який визначений вище). Такій структурі відповідає наступна схема граматики:

$$\Gamma_{10.21}: R = \{G \to HQ \qquad (6)$$
$$G \to (H)Q \qquad (7)$$
$$Q \to WG \qquad (8)$$
$$Q \to \$\}. \qquad (9)$$

Для побудови граматики арифметичних виразів, що допускають застосування вкладених дужок, варто передбачити можливість використання як елемента списку не тільки виразу без дужок, але і виразу, в якому можуть бути використані дужки. Схема граматики, що враховує цю можливість, може бути записана у вигляді:

$$\Gamma_{10.22}: \quad R = \{E \to GP \qquad (10)$$
$$E \to (E)P \qquad (11)$$
$$P \to WE \qquad (12)$$
$$P \to \$\}. \qquad (13)$$

Виведення ланцюжка $((i + i) * i)$ буде таким: $E \Rightarrow (E)P \Rightarrow (GP)P \Rightarrow ((H)Q\ P\ )P \Rightarrow ((iK)Q\ P\ )P \Rightarrow ((i\ W\ iK)Q\ P\ )P \Rightarrow ((i + iK)Q\ P\ )P \Rightarrow ((i + i)Q\ P\ )P \Rightarrow ((i + i)\ WG\ P\ )P \Rightarrow ((i + i)\ *G\ P\ )P \Rightarrow ((i + i)\ *\ HQP\ )P \Rightarrow ((i + i)*i\ KQP)P \Rightarrow ((i + i)\ *\ iQP)P \Rightarrow ((i + i)\ *\ iP)P \Rightarrow ((i + i)\ *\ i)P \Rightarrow ((i + i)\ *\ i)$ та матиме синтаксичний розбір [11, 10, 7, 1, 2, 4, 3, 8, 5, 6, 1, 3, 9, 13, 13]

10.7.6. Граматика для описів. Для наочності сприйняття надалі введемо допущення, згідно з яким ключові слова мови програмування будемо вважати одним термінальним символом. Нехай потрібно побудувати граматику для опису змінних цілого та дійсного типів. Для спрощення за змінні візьмемо тільки змінну *i*. Приклади ланцюжків, які описує дана граматика, наведені нижче:

1) int i, i, i, i; float i,i; int i; int i, i, i;

2) float i,i; int i, i, i; int i;

3) int i, i, i; float i, i; int i; float i, i; .

Тобто така граматика може генерувати ланцюжки, які містять послідовність списків змінних *i* типу *'float'* або *'int'*.

Перше правило граматики повинно починатися заданням типу *float* або *int* (нетермінальний символ *B*). Позначимо список змінних нетермінальним символом *C*. Виходячи з цього, маємо перше правило у вигляді $Z \to BC$. Структуру повного опису можна подати у вигляді двох вкладених списків з роздільниками. Внутрішній список (позначений нетермінальним символом *L*), розглянутий як елемент зовнішнього списку, являє собою опис змінних одного типу. Він має заголовок у вигляді покажчика типу, за яким випливає послідовність ідентифікаторів *i*, розділених комами. Зовнішній список (позначений нетермінальним символом *C*) використовує як роздільник крапку з комою.

Схема граматики розглянутого вигляду може бути записана так:

$$\Gamma_{10.23}: R = \{Z \to BC \quad (1)$$
$$C \to ;BC \quad (2)$$
$$C \to \$ \quad (3)$$
$$B \to real\ L \quad (4)$$
$$B \to int\ L \quad (5)$$
$$L \to iK \quad (6)$$
$$K \to ,iK \quad (7)$$
$$K \to \$\} \quad (8)$$

10.7.7. Граматика, що описує умовний оператор. Припустимо, що розглядається умовний оператор, аналогічний тому, який використовується в алгоритмічній мові Паскаль, з ключовими словами *if, then, else*. Як умову (нетермінальний символ *R)* в таких операторах дозволяється використовувати відносини, які складаються з ідентифікатора *a* та числа від одного до дев'яти (нетермінальний символ *D)*, з'єднаних знаками: =, >, <, <> (нетермінальний символ *Z)*. Структура такого оператора (нетермінальний символ *C)* може бути короткою (без *else*) та довгою. Вважаємо, наприклад, що після слова *then* слідує оператор *readln* (*F*), а після слова *else* – оператор *writeln*(*F*), де *F* є нетермінальним символом, який задає один із двох ідентифікаторів *b* або *c*. Умовний оператор закінчується крапкою з комою, а саме – ';'.

Приклади ланцюжків, які описує дана граматика, наведені нижче:

1) if a > 9 *then readln*(*b*);

2) *if a* = 5 *then readln*(*c*);

3) *if a* <> 0 *then readln*(*b*) *else writeln*(*c*).

Схема граматики, що задає ці послідовності, може бути зображена так:

$\Gamma_{10.24}$: R = {$I \rightarrow$ *if P A* (1)

$P \rightarrow$ *aZD* (2)

$Z \rightarrow$ < | > | <> | = (3)

$D \rightarrow$ 0 | 1 | 2 | … | 9 (4)

$A \rightarrow$ *then E C* (5)

$C \rightarrow$ *else K;* (6)

$C \rightarrow$ *;* (7)

$E \rightarrow$ *readln* (*F*) (8)

$K \rightarrow$ *writeln* (*F*) (9)

$F \rightarrow$ *b* | *c*} . (10)

Виведення ланцюжка *if a* < 0 *then writeln*(*c*)*;* буде таким: $I \Rightarrow$ *if PA* $\Rightarrow$ *if aZD A* $\Rightarrow$ *if a* < *DA* $\Rightarrow$ *if a* < 0 *A* $\Rightarrow$ *if a* < 0 *then EC* $\Rightarrow$ *if a* < 0 *then readln* (*F*) *C* $\Rightarrow$ *if a* < 0 *then readln* (*c*) *C* $\Rightarrow$ *if a* < 0 *then readln* (*c*)*;* та матиме синтаксичний розбір [1, 2, 3, 4.1, 5, 8, 10.2, 7].

Контрольні запитання

1. Дайте визначення формальної граматики.
2. Які символи граматики називаються термінальними, а які – нетермінальними?
3. Що називається мовою, яка породжується граматикою?
4. Які існують типи граматик?
5. Яка граматика називається контекстно-вільною граматикою?
6. Які граматики називаються неоднозначними?
7. Які існують способи задання схем граматик?
8. Що являє собою ітераційна форма?
9. Які існують рекомендації стосовно побудови правил граматики?
10. Побудувати правила граматики, які будуть описувати прототипи функцій.

11. КОНТЕКСТНО-ВІЛЬНІ ГРАМАТИКИ І АВТОМАТИ

11.1. Приведені граматики

З чотирьох типів граматик контекстно-вільні граматики (КВ) найбільш важливі з погляду додатків до мов програмування і компіляції. За допомогою КВ-граматики можна визначити велику частину структури мови програмування. При побудові граматик, що задають конструкції мов програмування, часто доводиться вдаватися до їх перетворення, щоб породжувана мова набула потрібної структури. Тому спочатку розглянемо трохи досить простих, але важливих перетворень КВ-граматик. Перший вид перетворення пов'язаний з видаленням із граматики зайвих символів. Зайві символи в граматиці можуть виявитися в таких випадках:

а) якщо символ не може бути отриманий при виведенні;

б) якщо із символу не може бути отриманий кінцевий термінальний ланцюжок (виходить нескінченний ланцюжок або немає правил, що приводять до термінального ланцюжка).

Спочатку розглянемо алгоритм виявлення символів, з яких не можна вивести кінцеві ланцюжки.

11.1.1. Визначення непродуктивних символів. Символ $X \in V_a$ називається **непродуктивним**, якщо з нього не може бути виведений кінцевий термінальний ланцюжок.

Розглядаючи правила граматики, можна зробити висновок, що, коли всі символи правої частини є продуктивними, то продуктивним є і символ, що стоїть у лівій частині. Останнє твердження дозволяє написати процедуру виявлення непродуктивних символів у такому вигляді:

1. Скласти список нетермінальних символів, для яких знайдеться хоча б одне правило, права частина якого містить термінальні символи або символ пусто (*$*).

2. Якщо знайдене таке правило і всі нетермінальні символи, які стоять у його правій частині, вже занесені до списку, то додати в список нетермінальний символ, що стоїть у його лівій частині.

3. Якщо на кроці 2 список більше не поповнюється, то отримано список усіх продуктивних нетермінальних символів граматики, а всі нетермінальні символи, які не потрапили в нього, є непродуктивними.

Приклад. Маємо граматику

$$\Gamma_{11.1}: \quad R = \{I \to aIa \quad (1)$$
$$I \to bAd \quad (2)$$
$$I \to c \quad (3)$$
$$A \to cBd \quad (4)$$
$$A \to aAd \quad (5)$$
$$B \to dAf\ \}. \quad (6)$$

На першому кроці заносимо до списку нетермінальний символ *I,* тому що права частина правила (3) не містить нетермінальних символів. На другому кроці, згідно з правилом (1), до списку потрібно б було занести нетермінальний символ *I*, але він там уже є. Тому наш список залишається без змін. Згідно з кроком три список не поповнюється, тому виявляється, що в заданій граматиці непродуктивними є символи *A* і *B*. Після виключення правил, які містять непродуктивні символи, одержуємо граматику:

$$R' = \{I \to a\, I\, a, \qquad (1)$$

$$I \to c\}. \qquad (2)$$

*11.1.2. Визначення недосяжних символів***.** У КВ-граматиці Г називається **недосяжним** символ $X \in V_т \cup V_а$, якщо X не з'являється в жодному виведеному ланцюжку.

Розглядаючи правила граматики, можна помітити, що, якщо нетермінальний символ у лівій частині правила є досяжним, то і всі символи правої частини є досяжними. Ця властивість правил є основою процедури виявлення недосяжних символів, яку можна описати так:

1. Створити одноелементний список, що складається з початкового символу граматики *I*.

2. Якщо знайдено правило, ліва частина якого вже є в списку, то включити до списку всі символи, які містяться в його правій частині.

3. Якщо на кроці 2 нові нетермінальні символи в список більше не додаються, то отримано список усіх досяжних нетермінальних символів, а решта нетермінальних символів, які не потрапили в список, є недосяжними.

Приклад. Маємо граматику

$$\Gamma_{11.2}: \quad R = \{I \to a\, I\, b \qquad (1)$$

$$I \to c \qquad (2)$$

$$A \to bI \qquad (3)$$

$$A \to a\}. \qquad (4)$$

На першому кроці до списку заносимо нетермінальний символ *I*. На другому кроці, згідно з правилом (2), потрібно додати до списку нетермінальний символ *I*, але він там уже є. Більше нетермінальних символів додати до списку ми не можемо. Отже, *A* є недосяжним символом.

КВ-граматика називається **приведеною**, якщо вона не містить зайвих (непродуктивних і недосяжних) символів.

11.2. Виключення ліворекурсивних правил

Ряд граматик потребують виключення ліворекурсивних та ланцюгових правил.

Правило вигляду $A \to \alpha A$, де $A \in V_а$, та $\alpha \in (V_т \cup V_а)$, називається **праворекурсивним**, а правило вигляду $A \to A\alpha$ – **ліворекурсивним.**

Для кожної КВ-граматики Г, що містить ліворекурсивні правила, можна побудувати еквівалентну граматику Г′, що не містить ліворекурсивних правил.

Щоб показати техніку перетворення, розглянемо приклад.

Приклад. Маємо граматику

$$\Gamma_{11.3}: \quad R = \{E \to E + T \mid T \qquad (1)$$
$$T \to T * F \mid F \qquad (2)$$
$$F \to (E) \mid a\}. \qquad (3)$$

Правила $E \to E + T \mid T$ перетворимо в правила $E \to T \mid TE'$ і $E' \to +T \mid +TE$, а правила $T \to T * F \mid F$ перетворимо в правила $T \to F \mid FT'$ і $T' \to *F \mid *FT'$.

У результаті одержимо граматику Г**,** що має схему:

$R' = \{ E \to T$	(1)	$T \to FT'$	(6)
$E \to TE'$	(2)	$T' \to *F$	(7)
$E' \to + T$	(3)	$T' \to * FT'$	(8)
$E' \to + TE'$	(4)	$F \to a$	(9)
$T \to F$	(5)	$F \to (E)\}.$	(10)

11.3. Виключення ланцюгових правил

Правило граматики вигляду $A \to B$, де $A, B \in V_a$, називається **ланцюговим.** Для КВ-граматики Г, яка містить ланцюгові правила, можна побудувати еквівалентну їй граматику Г′, яка не містить ланцюгових правил.

Ідея доведення полягає в наступному. Якщо схема граматики має вигляд

$$R = \{..., A \to B, ..., B \to C, ..., C \to aX\},$$

то така граматика еквівалентна граматиці зі схемою

$$R' = \{..., A \to aX, ...\},$$

оскільки виведення у граматиці зі схемою R ланцюжка aX

$$A \Rightarrow B \Rightarrow C \Rightarrow aX$$

може бути отримано у граматиці зі схемою R' за допомогою правила $A \to aX$.

11.4. Магазинні автомати

КВ-граматика визначає структуру ланцюжків і дозволяє будувати ланцюжки визначеної мови. Магазинні автомати (МА) дозволяють розв’язувати для контекстно-вільних мов задачу розпізнавання, яка полягає в тому, що за заданим ланцюжком необхідно визначити, чи належить він заданій мові. У роботах, пов'язаних з формальними мовами і граматиками, використовується модель магазинного автомата (рис 11.1), що складається з вхідної стрічки, управляючого пристрою і допоміжної стрічки, названої магазином або стеком. Вхідна стрічка розділяється на клітини (позиції), у кожній з яких може бути записаний символ вхідного алфавіту. При цьому передбачається, що у вільних клітинах вхідної стрічки розташовані порожні символи ε. Допоміжна стрічка також розділена на клітини, у яких можуть розташовуватися символи магазинного алфавіту. Початок допоміжної стрічки називається дном магазина та позначається – h_0. Зв'язок пристрою керування зі стрічками здійснюється двома головками, що можуть переміщатися уздовж стрічок.

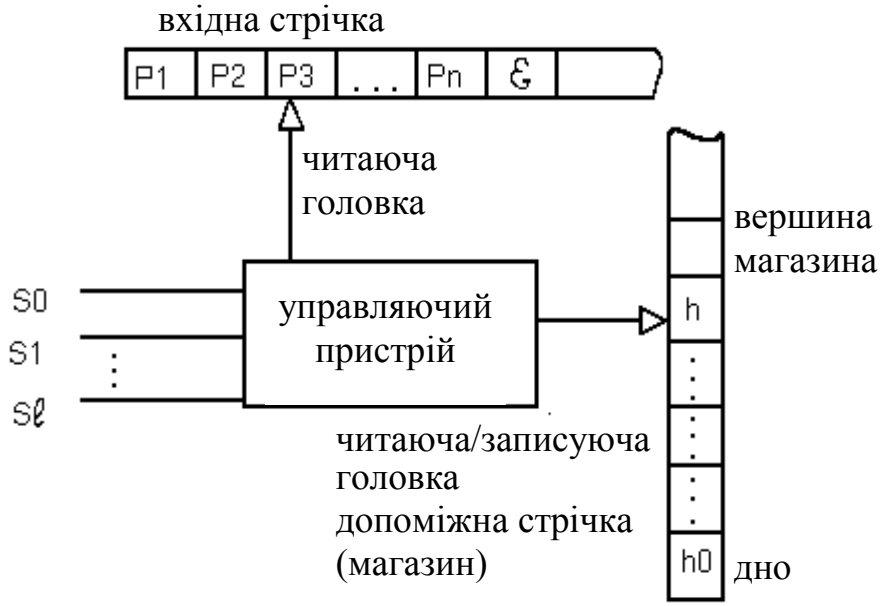


Рисунок 11.1 – Магазинний автомат

Головка вхідної стрічки може переміщатися тільки в один бік – вправо чи залишатися на місці. Вона може виконувати тільки операцію читання. Головка допоміжної стрічки здатна виконувати як читання, так і записування, але ці операції пов'язані з переміщенням головки певним чином:

- при записуванні головка попередньо зрушується на одну позицію вгору, а потім символ заноситься на стрічку;

- при читанні символ, що знаходиться під головкою, зчитується зі стрічки, а потім головка зрушується на одну позицію вниз, так що головка завжди встановлена напроти останнього записаного символу. Позицію, що знаходиться в розглянутий момент часу під головкою, називають **вершиною** магазина.

Магазинний автомат M визначається такою сукупністю семи об'єктів:

$M = \{P, S, s_o, f, F, H, h_o\}$, де P – вхідний алфавіт; S – алфавіт станів; s_o – початковий стан, $s_o \in S$; F – множина кінцевих станів, F є підмножиною S; H – алфавіт магазинних символів, які записуються на допоміжну стрічку; h_o – маркер дна, він завжди записується на дно магазина, $h_o \in H$; f – функція переходів.

Робота МА зводиться до зміни конфігурацій. Конфігурацією автомата називають трійку об'єктів (s, α, γ), де s – поточний стан автомата; α – вхідний ланцюжок, самий лівий символ a якого знаходиться під головкою, γ – магазинний ланцюжок, самий верхній символ якого $h \in H$. Якщо $a = \varepsilon$, то вважається, що вхідний ланцюжок прочитаний, якщо $\gamma = \$$ – то магазин порожній.

Робота автомата може бути подана як зміна конфігурацій. Один такт роботи автомата полягає у визначенні нової конфігурації за заданою. Це записується так:

$$(s, a\alpha, \gamma h) \text{ |-- } (s', \alpha, \gamma\beta).$$

При цьому передбачається, що автомат читає символ a, який знаходиться під головкою, і символ h, що знаходиться у вершині магазина, визначає новий стан s' і записує ланцюжок β у магазин замість символу h.

Якщо $\beta = \$$, то верхній символ виявляється вилученим з магазина. Така зміна конфігурацій можлива, якщо функція $f(s, a, h)$ визначена і їй належить значення (s', β). Описаний такт роботи виконується з переміщенням головки. Для опису роботи автомата також буде потрібен інший вид такту, що припускає зміну стану магазина без переміщення вхідної головки. Якщо $f^0(s, \varepsilon, h)$ визначена і їй належить значення (s', β), то зміна конфігурацій визначається так:

$$(s, a\alpha, \gamma h) \text{ |-- } (s', a\alpha, \gamma\beta).$$

У загальному вигляді дії, що задаються функцією переходів і виконуються автоматом, демонструє така форма запису:

f(*s*, <*вхідний символ*>, <*магазинний символ*>) = (s_1, < *ланцюжок, що заноситься до магазина*>).

При цьому варто мати на увазі, що при виконанні такту спочатку читається символ у вершині, а потім на його місце заноситься новий символ чи ланцюжок.

Початковою конфігурацією називається конфігурація (s_o, α, h_o), де s_o – початковий стан і h_o – маркер дна магазина, а **заключною** – конфігурація (s, $, $), де s належить до множини кінцевих станів F**.**

Для позначення послідовності змінюючих одна одну конфігурацій умовимося використовувати знак ├. У такий спосіб послідовність

$$(s_1, \alpha_1, \gamma_1) \vdash (s_2, \alpha_2, \gamma_2) \vdash \ldots \vdash (s_n, \alpha_n, \gamma_n)$$

записується в скороченому вигляді так:

$$(s_1, \alpha_1, \gamma_1) \vdash^* (s_n, \alpha_n, \gamma_n).$$

Для спрощення далі будемо використовувати єдиний стан s.

Контрольні запитання

1. Яка граматика називається приведеною?
2. Які символи граматики називаються непродуктивними?
3. Які символи граматики називаються недосяжними?
4. Які символи граматики називаються зайвими?
5. Які правила називаються ліворекурсивними?
6. Як можна виключити рекурсію?
7. Що таке магазинний автомат? Яка його структура?
8. Яка мова вважається допустимою для магазинного автомата?
9. Який автомат називається детермінованим, а який – недетермінованим?

12. СПАДНІ РОЗПІЗНАВАЧІ

Моделювання роботи недетермінованих **магазинних розпізнавачів** пов'язане з пошуком послідовності переходів з початкового в один з кінцевих станів. Пошук складається з окремих кроків, кожний з яких може закінчитися невдало і призвести до повернення у початковий стан для виконання наступного кроку. Такий пошук з поверненням пов'язаний зі значними витратами часу, тому на практиці використовують більш економічні детерміновані розпізнавачі, що працюють без повернень. Ці розпізнавачі допускають тільки обмежені класи контекстно-вільних мов, які відображають усі синтаксичні риси мов програмування.

Розпізнавачі можна розділити на дві категорії: **спадні (нисхідні) та висхідні**. Кожна категорія характеризується порядком, в якому розташовуються правила в дереві виведення. **Спадні розпізнавачі** обробляють правила зверху вниз, тобто – верхні правила раніш нижніх, у той час як висхідні розпізнавачі використовують нижні правила раніш тих, які розташовані вище.

У загальному випадку граматика належить до класу $LL(K)$-граматик, якщо для неї можна побудувати спадний детермінований розпізнавач, що враховує K вхідних символів, розташованих праворуч від поточної вхідної позиції.

Назва *LL* походить від слова *Left*, оскільки розпізнавач переглядає вхідний ланцюжок ліворуч – праворуч, і від слова *Leftmost*, оскільки він виявляє появу правила за одним чи групою символів, що утворюють лівий край ланцюжка. На практиці найбільше застосовується клас $LL(1)$-граматик, для яких детермінований розпізнавач працює за одним вхідним символом, розташованим в поточній позиції. До класу $LL(1)$-граматик належать також розділені та слаборозділені граматики.

12.1. Розділені граматики

Контекстно-вільна граматика, яка не містить правил, що анулюють, називається **розділеною**, або простою, якщо виконуються такі дві умови:

1. Права частина кожного правила починається термінальним символом.
2. Якщо два правила мають однакові ліві частини, то праві частини цих правил повинні починатися різними термінальними символами.

Важливою властивістю розділених граматик є те, що для кожної з них можна побудувати **детермінований спадний розпізнавач.**

12.2. Побудова детермінованого спадного розпізнавача

Побудова розпізнавача передбачає протиставлення кожному правилу граматики команди розпізнавача. Відповідно до загального способу побудови розпізнавача кожному правилу розділеної граматики, що має вигляд

$$A \to a\,\alpha,$$

де α – ланцюжок символів повного словника і a належить термінальному словнику, потрібно поставити у відповідність команду

$$f(s, a, A) = (s, \alpha'),$$

яка визначає такт роботи розпізнавача зі зрушенням вхідної головки і у якій α' являє собою дзеркальне відображення ланцюжка α.

Крім того, варто врахувати, що термінальні символи можуть бути розташовані в правих частинах правил не тільки на самій лівій позиції. Для таких термінальних символів необхідно побудувати команди вигляду

$$f(s, b, b) = (s, \$).$$

Для переходу в заключний стан додамо правило

$$f(s, \$, h_0) = (s, \$),$$

а за початкову конфігурацію розпізнавача приймемо вираз

$$(s, \alpha, h_0 I),$$

де I – початковий символ граматики, а α – заданий вхідний ланцюжок.

Застосовуючи наведені вище правила, побудуємо розпізнавач для розділеної граматики $\Gamma_{12.1}$:

$$\Gamma_{12.1}: \; R = \{I \to abB,\; I \to bBbI,\; B \to a,\; B \to bB\},$$

$$P = \{a, b\},\; H = \{a, b, I, B, h_0\},\; S = \{s\},\; F = \{s\}.$$

Побудуємо команди розпізнавача:

$$f(s, a, I) = (s, Bb) \quad (1)$$
$$f(s, a, B) = (s, \$) \quad (2)$$
$$f(s, b, I) = (s, IbB) \quad (3)$$
$$f(s, b, B) = (s, B) \quad (4)$$
$$f(s, b, b) = (s, \$) \quad (5)$$
$$f(s, \$, h_0) = (s, \$) \quad . \quad (6)$$

Роботу побудованого автомата покажемо на прикладі аналізу ланцюжка *bbababa*:

$(s, bbababa, h_0I) \vdash_3 (s, bababa, h_0IbB) \vdash_4 (s, ababa, h_0IbB) \vdash_2 (s, baba, h_0Ib) \vdash_5 (s, aba, h_0I) \vdash_3 (s, ba, h_0Bb) \vdash_5 (s, a, h_0B) \vdash_2 (s, \$, h_0) \vdash_6 (s, \$, \$)$.

Послідовність команд, яка дозволяє виконати вищенаведену зміну конфігурацій, є такою [3, 4, 2, 5, 3, 5, 2, 6].

Наведена послідовність конфігурацій показує, що в кожній конфігурації може бути застосована єдина команда розпізнавача, тому такий розпізнавач називається **детермінованим**.

12.2.1. Множина ВИБІР. Інший клас граматик, який породжують детерміновані мови, називається класом **слаборозділених граматик**. Цей клас відрізняється від класу розділених граматик тим, що він допускає використання правил, які анулюють, у схемі граматики. Однак не всяка розділена граматика з правилами, що анулюють, належить до класу слаборозділених граматик. Щоб сформулювати визначення, яке дозволить розпізнавати слаборозділені і *LL*(1)

граматики, нам будуть потрібні нові поняття: множина **ВИБІР** та функції **ПЕРШ(*μ*)** і **СЛІД(*A*)** .

Множина ВИБІР будується для кожного правила і включає ті термінальні символи, з появою яких під читаючою головкою розпізнавач має застосувати відповідне правило.

Для визначення множини ВИБІР використовуються функції ПЕРШ(μ) і СЛІД(A). Аргументом функції ПЕРШ(μ) може бути будь-який ланцюжок повного словника μ, а значенням функції ПЕРШ(μ) є множина термінальних символів, що можуть стояти на першому місці в ланцюжках, виведених з ланцюжка μ.

12.2.2. Побудова функції ПЕРШ(μ). Значення функції ПЕРШ(μ) можна визначити, користуючись такими правилами:

1) Якщо ланцюжок μ починається термінальним символом і має вигляд $\mu \rightarrow b\mu'$, то функція

$$\text{ПЕРШ}(A \rightarrow b\mu') = \{b\}.$$

2) Якщо ланцюжок μ є порожнім, тобто $\mu \rightarrow \$$, то функція

$$\text{ПЕРШ}(A \rightarrow \$) = \{\$\}.$$

3) Якщо ланцюжок μ починається нетермінальним символом B і має вигляд $\mu \rightarrow B\mu'$, а в схемі граматики є n правил, у будь-якій частині яких знаходиться символ B:

$$B \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n,$$

і якщо не існує виведення $B \rightarrow \$$, то функція ПЕРШ($\mu \rightarrow B\mu'$) являє собою об'єднання множин:

$$\text{ПЕРШ}(\mu \rightarrow B\mu') = \text{ПЕРШ}(\alpha_1) \cup \text{ПЕРШ}(\alpha_2) \cup \ldots \cup \text{ПЕРШ}(\alpha_n).$$

4) Якщо ланцюжок μ починається нетермінальним символом і має вигляд $\mu \rightarrow B\mu'$, а в схему граматики входять n правил вигляду

$$B \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n,$$

і B є нетермінальним символом, що анулює, тобто існує $B \rightarrow \$$, то функція

$$\text{ПЕРШ}(\mu \rightarrow B\mu') = \text{ПЕРШ}(\mu') \cup \text{ПЕРШ}(\alpha_1) \cup \text{ПЕРШ}(\alpha_2) \cup \ldots \cup \text{ПЕРШ}(\alpha_n).$$

Як приклад виконаємо обчислення функції ПЕРШ(μ) для правил граматики $\Gamma_{12.2}$:

$\Gamma_{12..2}$: $R = \{$

$I \rightarrow AB$;	(1)
$A \rightarrow int$	(2)
$A \rightarrow float$	(3)
$A \rightarrow char$	(4)
$B \rightarrow CDE$	(5)
$C \rightarrow *$	(6)
$C \rightarrow \$$	(7)
$D \rightarrow i$	(8)
$D \rightarrow a$	(9)
$E \rightarrow ,CDE$	(10)
$E \rightarrow \$\}$.	(11)

Спочатку знайдемо значення функції ПЕРШ(μ) для правил, права частина яких починається з термінального символу або з символу *$* (правила з номерами 2, 3, 4, 6, 7, 8, 9, 10, 11).

ПЕРШ($A \rightarrow int$) = {*int*}
ПЕРШ($A \rightarrow float$) = {*float*}
ПЕРШ($A \rightarrow char$) = {*char*}

Об'єднавши три значення, отримаємо ПЕРШ(A) = {*int*, *float*, *char*}

ПЕРШ($C \rightarrow$ *) = {*}
ПЕРШ($C \rightarrow \$$) = {*$*}

Об'єднавши значення, отримаємо ПЕРШ(C) = {*, *$*}

ПЕРШ($D \rightarrow i$) = {*i*}
ПЕРШ($D \rightarrow a$) = {*a*}

Об'єднавши значення, отримаємо ПЕРШ(D) = {*i*, *a*}

ПЕРШ($E \rightarrow$, $C\ D\ E$) = {,}
ПЕРШ($E \rightarrow \$$) = {*$*}

Об'єднавши значення, отримаємо ПЕРШ(E) = {, *$*}

Знаходимо функцію ПЕРШ(μ) для першого правила:

ПЕРШ($I \rightarrow A\ B$;) = ПЕРШ(A) = {*int*, *float*, *char*}.

Функція ПЕРШ для п'ятого правила знаходиться складніше. Оскільки перший нетермінальний символ C правої частини правила згідно з сьомим правилом може бути анулюючим, то до складу функції, окрім ПЕРШ(C), буде входити також ПЕРШ(D).

ПЕРШ($B \rightarrow CDE$) = ПЕРШ(C) $\cup$ ПЕРШ(D) = {*, *$*} $\cup$ {*i*, *a*} = {*, *i*, *a*}.

Розглянемо обчислення функції ПЕРШ для правил більш складнішої граматики $\Gamma_{12.3}$:

$\Gamma_{12.3}$: R = { $A \rightarrow BCc$ (1); $C \rightarrow ca$ (6)
$A \rightarrow gDB$ (2); $D \rightarrow \$$ (7)
$B \rightarrow \$$ (3); $D \rightarrow dD$ (8)
$B \rightarrow bCDE$ (4); $E \rightarrow gAf$ (9)
$C \rightarrow DaB$ (5); $E \rightarrow c$ }. (10)

Спочатку знайдемо значення функції для правих частин правил (2), (4), (6), (8), (9), (10), що починаються термінальними символами:

ПЕРШ($A \rightarrow gDB$) = {*g*}
ПЕРШ($B \rightarrow bCDE$) = {*b*}
ПЕРШ($C \rightarrow ca$) = {*c*}
ПЕРШ($D \rightarrow dD$) = {*d*}
ПЕРШ($E \rightarrow gAf$) = {*g*}
ПЕРШ($E \rightarrow c$) = {*c*}.

Потім обчислимо функцію для правил (5) і (6):

ПЕРШ(C) = ПЕРШ(D) $\cup$ ПЕРШ(c).

З огляду на те, що D є нетермінальним символом, що анулює, одержуємо:

ПЕРШ(C) = ПЕРШ(a) $\cup$ ПЕРШ(D) $\cup$ {*c*} = {*a*} $\cup$ {*d*} $\cup$ {*c*} = {*a*, *d*, *c*}.

При обчисленні функції для правил (1) і (2) також необхідно врахувати те, що B є нетермінальним символом, що анулює, тому маємо:

ПЕРШ(A) = ПЕРШ(B) ∪ ПЕРШ(g) = ПЕРШ(C) ∪ ПЕРШ(B) ∪ ПЕРШ(g) =
= $\{a, d, c\} \cup \{b\} \cup \{g\} = \{a, b, c, d, g\}$.

12.2.3. Побудова функції СЛІД(B). Аргументом функції СЛІД(В) є нетермінальний символ, наприклад B, а значення функції СЛІД(В) являє собою множину термінальних символів, що можуть знаходитись безпосередньо за нетермінальним символом B у ланцюжках, виведених з початкового символу граматики.

Обчислення значення функції СЛІД(B) повинне виконуватися за такими правилами:

1) Якщо в схемі граматики є правила вигляду

$$X_1 \to \mu_1 B\, \alpha_1, \quad X_2 \to \mu_2 B\, \alpha_2, \ldots, X_n \to \mu_n B\, \alpha_n,$$

і не існує правил $\alpha_i \to \$$, то

СЛІД(B) = ПЕРШ(α_1) ∪ ПЕРШ(α_2) ∪ ... ∪ ПЕРШ(α_n).

2) Якщо ж серед наведених вище правил існує хоча б один ланцюжок $\alpha_i \to \$$, (наприклад, нехай існує правило $X_1 \to \mu_1 B$ тобто $\alpha_1 \to \$$), то функція обчислюється так:

СЛІД(B) = СЛІД(X_1) ∪ ПЕРШ(α_2) ∪ ... ∪ ПЕРШ(α_n).

Виконаємо обчислення функції СЛІД для нетермінальних символів граматики $\Gamma_{12.2}$:

СЛІД(A) = ПЕРШ(B) = {*, i, a}
СЛІД(B) = СЛІД(I) = {$}
СЛІД(C) = ПЕРШ(D) = {i, a}
СЛІД(D) = ПЕРШ(E) = {, , $}
СЛІД(E) =СЛІД(B) ∪ СЛІД(E) = СЛІД(I) ∪ {$} = {$}.

Виконаємо обчислення функції СЛІД для нетермінальних символів граматики $\Gamma_{12.3.}$ Спочатку визначимо функцію для нетермінального символу A, що зустрічається в правій частині правила (9).

СЛІД(A) = ПЕРШ(f) = $\{f\}$.

Нетермінальний символ C належить до правої частини правил (1) і (4). З огляду на те, що нетермінальний символ D є анулюючим, одержуємо

СЛІД(C) = ПЕРШ(D) ∪ ПЕРШ(E) ∪ ПЕРШ(c) = $\{c, d, g\}$.

Нетермінальний символ B належить до правої частини правил (1), (2) і (5). Тому маємо

СЛІД(B) = ПЕРШ(Cc) ∪ СЛІД(A) ∪ СЛІД(C).

Підставляючи в отриманий вираз значення функцій, що входять у праву частину, одержуємо

СЛІД(B) = $\{a, c, d\} \cup \{f\} \cup \{c, d, g\} = \{a, c, d, g, f\}$.

Для нетермінального символу D, що входить у праву частину правил (2), (4), (5) і (8), враховуючи те, що нетермінальний символ B є анулюючим, одержуємо

СЛІД(D) = ПЕРШ(B) ∪ СЛІД(A) ∪ ПЕРШ(E) ∪ ПЕРШ(aB).

З огляду на те, що для нетермінального символу E, який належить до правила (4),

$$\text{СЛІД}(E) = \text{СЛІД}(B) = \{a, d, c, g, f\},$$

остаточно маємо

$$\text{СЛІД}(D) = \text{ПЕРШ}(B) \cup \text{СЛІД}(A) \cup \text{ПЕРШ}(E) \cup \{a\} =$$
$$= \{b\} \cup \{f\} \cup \{c, g\} \cup \{a\} = \{a, b, c, g, f\}.$$

12.2.4. Побудова множини ВИБІР. Множину **ВИБІР** можна визначити за допомогою функцій ПЕРШ (μ) і СЛІД(В) у такий спосіб:

1) Якщо правило граматики має вигляд $B \to \alpha$ і α не є анулюючим ланцюжком, іншими словами, якщо не існує виведення $\alpha \to \$$, то

$$\text{ВИБІР}(B \to \alpha) = \text{ПЕРШ}(\alpha).$$

2) Для правил граматики, що анулюють, вигляду $B \to \$$ множина ВИБІР визначається так: $\text{ВИБІР}(B \to \$) = \text{СЛІД}(B)$.

3) Якщо правило граматики має вигляд $B \to \mu$ і μ є ланцюжком, що анулює, то

$$\text{ВИБІР}(B \to \mu) = \text{ПЕРШ}(\mu) \cup \text{СЛІД}(B).$$

Для розглянутої граматики $\Gamma_{12.2}$ множина ВИБІР для кожного з правил, побудованих описаним вище способом, має вигляд:

$\text{ВИБІР}(I \to AB;) = \text{ПЕРШ}(A) = \{int, char, float\}$
$\text{ВИБІР}(A \to int) = \{int\}$
$\text{ВИБІР}(A \to float) = \{float\}$
$\text{ВИБІР}(A \to char) = \{char\}$
$\text{ВИБІР}(B \to CDE) = \text{ПЕРШ}(C) \cup \text{ПЕРШ}(D) = \{*, i, a\}$
$\text{ВИБІР}(C \to *) = \{*\}$
$\text{ВИБІР}(C \to \$) = \text{СЛІД}(C) = \{i, a\}$
$\text{ВИБІР}(D \to i) = \{i\}$
$\text{ВИБІР}(D \to a) = \{a\}$
$\text{ВИБІР}(E \to , CDE) = \{,\}$
$\text{ВИБІР}(E \to \$) = \text{СЛІД}(E) = \{\$\}$.

Для розглянутої граматики $\Gamma_{12.3}$ множина ВИБІР для кожного з правил, побудованих описаним вище способом, має вигляд:

$\text{ВИБІР}(A \to BCc) = \text{ПЕРШ}(BCc) = \{a, b, c, d\}$,
$\text{ВИБІР}(A \to gDB) = \text{ПЕРШ}(gDB) = \{g\}$,
$\text{ВИБІР}(B \to \$) = \text{ПЕРШ}(\$) \cup \text{СЛІД}(B) = \{a, c, d, g, f\}$,
$\text{ВИБІР}(B \to bCDE) = \text{ПЕРШ}(bCDE) = \{b\}$,
$\text{ВИБІР}(C \to DaB) = \text{ПЕРШ}(DaB) = \{a, d\}$,
$\text{ВИБІР}(C \to ca) = \text{ПЕРШ}(ca) = \{c\}$,
$\text{ВИБІР}(D \to \$) = \text{ПЕРШ}(\$) \cup \text{СЛІД}(D) = \{a, b, c, g, f\}$,
$\text{ВИБІР}(D \to dD) = \text{ПЕРШ}(dD) = \{d\}$,
$\text{ВИБІР}(E \to gAf) = \text{ПЕРШ}(gAf) = \{g\}$,
$\text{ВИБІР}(E \to c) = \text{ПЕРШ}(c) = \{c\}$.

12.3. Слаборозділені граматики

Використовуючи введені поняття, можна дати визначення слаборозділеної граматики.

КВ-граматика називається **слаборозділеною**, якщо виконуються такі три умови:

- права частина кожного правила являє собою або порожній ланцюжок *$,* або починається з термінального символу;
- якщо два правила мають однакові ліві частини, то праві частини правил повинні починатися різними символами;
- множина ВИБІР, побудована для правил з однаковою лівою частиною, не містить однакових елементів.

12.4. *LL*(1)-граматики

Розділені і слаборозділені граматики є підкласом граматик більш загального виду, що називаються *LL*(1)-граматиками. Граматика називається *LL*(1)-граматикою за умов:

- права частина кожного правила являє собою або порожній ланцюжок *$,* або починається з термінального чи нетермінального символу;
- якщо два правила мають однакові ліві частини, то праві частини правил повинні починатися різними символами;
- множина ВИБІР, побудована для правил з однаковою лівою частиною, не містить однакових елементів.

12.5. Побудова магазинного автомата

Для граматик, що відповідають умовам *LL*(1)-граматик, правдиве наступне твердження: для кожної *LL*(1)-граматики можна побудувати детермінований магазинний автомат М, що допускає мову, породжувану даною граматикою:

$$L(\Gamma) = L(\text{M}).$$

Задача побудови магазинного автомата для заданої *LL*(1)-граматики формулюється таким чином.

Задано граматику $\Gamma = \{V_т, V_a, I, R\}$, потрібно визначити об'єкти, що визначають автомат $\text{M} = \{P, S, s, F, H, h_0, f\}$.

З огляду на те, що автомат повинен допускати ланцюжки мови, породжуваної заданою граматикою, приймемо $P = V_т$. Щоб спростити опис автомата, припустимо, що він має **один стан *s***, який є і початковим і заключним, тобто – $S = \{s\}$ і $F = \{s\}$.

Як магазинний алфавіт приймемо таке об'єднання:

$$H = \{V_т \cup V_а \cup \{h_0\}\}.$$

Побудову функцій переходів виконаємо з використанням множин ВИБІР правил заданої граматики:

1) Для кожного правила граматики, що починається термінальним символом вигляду $A \to b\alpha$, побудуємо команду автомата

$$f(s, b, A) = (s, \alpha'),$$

де α' є дзеркальним відображенням ланцюжка α.

2) Для кожного правила граматики, що починається нетермінальним символом вигляду $A \to B\alpha$ побудуємо команду автомата

$$f^*(s, x, A) = (s, B\alpha'),$$

де f^* – команда автомата без зрушення вхідної головки, α' – дзеркальне відображення ланцюжка α, x – елемент множини ВИБІР. Кількість команд, які необхідно побудувати для заданого правила, визначається числом елементів множини ВИБІР для цього правила. Команди магазинного автомата, що працюють без зрушення вхідної головки, виконують заміну нетермінального символу, який відповідає лівій частині правила, дзеркальним відображенням ланцюжка, який відповідає правій частини цього правила.

3) Для кожного правила граматики, що анулює ($A \to \$$), побудуємо команди автомата без зрушення вхідної головки

$$f^*(s, x, A) = (s, \$),$$

де x – елемент множини ВИБІР($A \to \$$). Кількість таких команд визначається числом елементів множини ВИБІР для цього правила.

4) Для кожного термінального символу b, розташованого в середині або на кінці правих частин правил граматики, побудуємо команду

$$f(s, b, b) = (s_0, \$).$$

5) Для переходу в заключний стан побудуємо команду

$$f^*(s, \$, h_0) = (s_0, \$, \$).$$

Як початкову конфігурацію автомата умовимося використовувати такий вираз з заданим вхідним ланцюжком μ:

$$(s, \mu, h_0 I).$$

12.6. Приклади побудови спадного розпізнавача

Приклад 1. Побудувати спадний розпізнавач для оператора присвоювання арифметичного виразу. Арифметичний вираз містить: *ідентифікатор i*, :=, (,), ;. Кількість вкладених дужок не обмежена.

Приклади ланцюжків оператора присвоювання:

$$i := i+(i+i)+(i+i);$$
$$i := i+i;$$
$$i := i+(i+(i+i)).$$

1. Будуємо правила граматики. Праву частину оператора присвоювання позначимо нетермінальним символом *A*. Перше правило матиме вигляд: $I \to i := A;$. Нетермінальний символ *A* є списком з роздільником. Елементом списку є ідентифікатор *i,* а роздільником є знак *«+»*. Окрім того, список *A* може бути

розміщеним у дужках. Нетермінальний символ C використовуємо для рекурсії.

Таким чином, маємо:

$$I \to i := A; \quad (1)$$
$$A \to iC \quad (2)$$
$$C \to +A \quad (3)$$
$$A \to (A)C \quad (4)$$
$$C \to \$. \quad (5)$$

2. Визначаємо, чи містить наша граматика непродуктивні символи, а саме знайдемо правило, в правій частині якого розташовано термінальний символ або символ $\$$. $\{C\}$ – продуктивний символ. Знайдемо правила, в правій частині котрих містяться символи, що вже занесені в список. Якщо такі правила існують, то додаємо в список нетермінальні символи, які розміщені в лівій частині правила:

$\{C\ A\}$
$\{C\ A\}$
$\{C\ A\ I\}$

Таким чином, у список потрапили всі нетермінальні символи, отже непродуктивних символів немає.

3. Шукаємо недосяжні символи.

Заносимо в список початковий символ граматики $\{I\}$.

Шукаємо правило, ліва частина якого вже занесена в список. Додаємо нетермінальні символи правої частини:

$\{I\ A\}$
$\{I\ A\ C\}$

У список потрапили всі нетермінальні символи, отже, недосяжних символів немає.

4. Дана граматика містить правила, що анулюють, та правила, права частина котрих починається з нетермінального символу, отже, вона не може бути розділеною та слаборозділеною граматикою. Визначимо належність даної граматики до $LL(1)$-граматики.

Аналізуючи складені нами правила, дійдемо висновку, що ліворекурсивних правил немає.

5. Побудуємо функцію ПЕРШ:

ПЕРШ $(I \to i := A;) = \{i\}$
ПЕРШ $(A \to iC) = \{i\}$
ПЕРШ $(C \to +A) = \{+\}$
ПЕРШ $(A \to (A)C) = \{(\}$
ПЕРШ $(C \to \$) = \{\$\}$.

6. Побудуємо функцію СЛІД:

Згідно з першим правилом СЛІД$(A) = \{;\}$. Згідно з чевертим правилом СЛІД$(A) = \{)\}$. Згідно з другим правилом СЛІД(C) = СЛІД (A). Згідно з третім правилом СЛІД(A) = СЛІД (C). Має місце зацикленість. Отже, СЛІД(A) = СЛІД

$(C) = \{;\} \cup \{)\} = \{;,)\}$.

7. Побудуємо множину ВИБІР. Для всіх правил, крім правила, що анулює, множина ВИБІР дорівнює функції ПЕРШ. Для правила, що анулює, множина ВИБІР дорівнює функції СЛІД.

ВИБІР($I \to i:= A;$) = ПЕРШ ($I \to i:= A;$) = $\{i\}$

ВИБІР($A \to iC$) = ПЕРШ ($A \to iC$) = $\{i\}$

ВИБІР($C \to +A$) = ПЕРШ ($C \to +A$) =$\{+\}$

ВИБІР($A \to (A)C$) = ПЕРШ ($A \to (A)C$) = $\{(\}$

ВИБІР($C \to \$$) = СЛІД(C) $\cup$ $\{\$\}$ = СЛІД(A) $\cup$ $\{\$\}$ = $\{;,)\}$

Дана граматика є $LL(1)$-граматикою, тому що для правил, які починаються з однакових нетермінальних символів, множина ВИБІР не містить однакових термінальних символів.

8. Будуємо команди магазинного автомата (розпізнавача):

1) $f(s, i, I) = (s, ;A:=)$
2) $f(s, i, A) = (s, C)$
3) $f(s, +, C) = (s, A)$
4) $f(s, (, A) = (s, C)A)$
5) $f^*(s, ;, C) = (s, \$)$
6) $f^*(s,), C) = (s, \$)$
7) $f(s, ;, ;) = (s, \$)$
8) $f(s,),)) = (s, \$)$
9) $f(s, :=, :=) = (s, \$)$
10) $f(s, \$, h_0) = (s, \$)$

Кількість команд для кожного правила визначається числом елементів множини ВИБІР. Оскільки п’яте правило є правилом, що анулює, то команди магазинного автомата 5 – 6 виконуються без зрушення вхідної головки. Команди 8 – 9 будуються для термінальних символів « ; », «) » та « := », які розташовані в середині та в кінці правил. Команда 10 використовується для переходу в заключний стан.

9. Розпізнаємо ланцюжок $i:=i+(i+i)$; відповідно до побудованих правил:

$(s,$	$i:=i+(i+i);\$,$	$h_0I)$ — 1
$(s,$	$:=i+(i+i);\$,$	$h_0;A:=)$ — 10
$(s,$	$i+(i+i);\$,$	$h_0;A)$ — 2
$(s,$	$+(i+i);\$,$	$h_0;C)$ — 3
$(s,$	$(i+i);\$,$	$h_0;A)$ — 4
$(s,$	$i+i);\$,$	$h_0;C)A)$ — 2
$(s,$	$+i);\$,$	$h_0;C)C)$ — 3
$(s,$	$i);\$,$	$h_0;C)A)$ — 2
$(s,$	$);\$,$	$h_0;C)C)$ — 6
$(s,$	$);\$,$	$h_0;C))$ — 9
$(s,$	$;\$,$	$h_0;C)$ — 5
$(s,$	$;\$,$	$h_0;)$ — 8
$(s,$	$\$,$	$h_0)$ — 11
$(s,$	$\$,$	$\$)$

Ланцюжок розпізнано.

Приклад 2. Побудувати спадний розпізнавач для оператора виведення *write* або *writeln*. Оператор може містити в дужках список ідентифікаторів *i* та коментарів ‘*t*‘.

Приклади ланцюжків:

write(‘*t*’,*i);*

writeln(‘*t*’,*i*, *i*);

write(*i*, *i*).

1. Будуємо правила граматики. Нетермінальним символом *A* позначимо частину ланцюжка, яка знаходиться після термінальних символів *write* або *writeln.* Нетермінальним символом *B* позначимо список операторів *i* та коментарів ‘*t*‘ з роздільником «,». Нетермінальний символ *C* використаємо для рекурсії.

$I \to write\ A;$ (1)

$I \to writeln\ A;$ (2)

$A \to (B)$ (3)

$A \to \$$ (4)

$B \to \text{'}t\text{'}C$ (5)

$B \to iC$ (6)

$C \to , B$ (7)

$C \to \$$ (8)

2. Дана граматика не містить марних символів та ліворекурсивних правил.

3. Дана граматика містить правила, що анулюють, та правила, права частина яких починається з нетермінального символу, отже, вона не може бути розділеною та слаборозділеною граматикою. Визначимо належність даної граматики до *LL*(1)-граматики.

ВИБІР($I \to write\ A;$) = ПЕРШ($I \to write\ A;$) = {*write*}

ВИБІР($I \to writelnA;$) = ПЕРШ($I \to writelnA;$) = {*writeln*}

ВИБІР($A \to (B)$) = ПЕРШ ($A \to (B)$) = {*(*}

ВИБІР($A \to \$$) = СЛІД(*A*) = {$,;}

ВИБІР($B \to \text{'}t\text{'}C$) = ПЕРШ ($B \to \text{'}t\text{'}C$) = { ‘}

ВИБІР($B \to iC$) = ПЕРШ($B \to iC$) = {*i*}

ВИБІР($C \to , B$) = ПЕРШ($C \to , B$) = {,}

ВИБІР($C \to \$$) = СЛІД(*C*) = СЛІД(*B*) = {*),$*}

Дана граматика є *LL*(1)-граматикою, тому що множина ВИБІР для правил, які починаються з однакових лівих нетермінальних символів, не містить однакових термінальних символів.

4. Будуємо команди розпізнавача:

1) $f(s, write, I) = (s, h_0;A)$
2) $f(s, writeln, I) = (s, h_0;A)$
3) $f(s, (, A) = (s, h_0)B)$
4) $f^*(s, ;, A) = (s, \$)$
5) $f^*(s, \$, A) = (s, \$)$

6) $f(s, ‘, B) = (s, C’t)$
7) $f(s, i, B) = (s, C)$
8) $f(s, , , C) = (s, B)$
9) $f *(s,),C) = (s, \$)$
10) $f *(s, \$, C) = (s, \$)$
11) $f(s, ; , ;) = (s, \$)$
12) $f(s,) ,)) = (s, \$)$
13) $f(s, ’, ‘) = (s, \$)$
14) $f(s, t, t) = (s, \$)$
15) $f(s, \$, h_0) = (\$, \$)$

5. Розпізнаємо заданий ланцюжок:

(s,	*writeln(‘t’,i, i);$*	*h0I)*	*1*
(s,	*(‘t’,i,i);$*	*h0;A)*	*3*
(s,	*‘t’,i,i); $*	*h0;)B)*	*6*
(s,	*t’,i,i);$*	*h0;)C’t)*	*14*
(s,	*’,i,i);$*	*h0;)C’)*	*13*
(s,	*,i,i);$*	*h0;)C)*	*8*
(s,	*i,i);$*	*h0;)B)*	*7*
(s,	*,i);$*	*h0;)C)*	*8*
(s,	*i);$,*	*h0;)B)*	*7*
(s,	*);$,*	*h0;)C)*	*9*
(s,	*);$,*	*h0;))*	*12*
(s,	*; $,*	*h0;)*	*11*
(s,	*$,*	*h0)*	*15*
($,	*$,*	*$)*	

Ланцюжок розпізнано.

Контрольні запитання

1. Як працює спадний розпізнавач?
2. Яка граматика називається розділеною або простою?
3. Які типи команд розпізнавача можуть бути зіставлені з кожним правилом граматики?
4. Яка граматика називається слаборозділеною?
5. Як будується функція ПЕРШ(μ)?
6. Як будується функція СЛІД(μ)?
7. Як будується множина ВИБІР($B \to \mu$)?
8. Які граматики називаються $LL(K)$–граматиками?
9. Як будуються команди розпізнавача для $LL(K)$–граматик?
10. Побудувати спадний розпізнавач для фрагмента програми, яка описує список змінних цілого і символьного типів.

СПИСОК ЛІТЕРАТУРИ

1. О.Е. Акимов. Дискретная математика: логика, группы, графы/ О.Е. Акимов. – М.: Лаборатория Базовых Знаний, 2001.
2. О.П. Кузнєцов. Дискретная математика для инженера/ О.П Кузнєцов., Д.М. Адельсон-Вельский. – 2-е изд. – М: Энергоатомиздат, 1988. – 488 с.
3. Ю.М. Коршунов Математические основы кибернетики/ Ю.М. Коршунов: учебное пособ. для втузов. М.: Энергия,1980. 376с.
4. Ф.А. Новиков Дискретная математика для программистов/ Ф.А. Новиков: учебник. – СПб: Питер, 2000. – 304 с.
5. Е.П.Липатов. Теория графов и ее применения/ Е.П Липатов. – М.: Знание, 1986. – 32 с.
6. И.Б. Сироджа. Комбинаторные задачи оптимизации на графах: Учебное пособ. по курсу "Спецглавы математики". – Харьков: ХАИ, 1991. – 84 с.
7. В.А. Евстигнеев. Применение теории графов в программировании/ В. А. Евстигнеев. – М.: Наука, 1985. – 200 с.
8. В.Н. Нефедов. Курс дискретной математики/ В.Н Нефедов, В.А. Осипова. – М.: Наука, 1992.
9. Л.М. Лихтарникова. Математическая логика: Курс лекций. Задачник-практикум и решения/ Л.М. Лихтарникова, Т.Г. Сукачева. – СПб, 1988.
10. С.Г. Гиндикин. Алгебра логики в задачах/ С.Г. Гиндикин. – М., 1972.
11. Н. Кристофидес. Теория графов. Алгоритмический поход/ Н. Кристофидес. – М.: Мир, 1978.
12. Р.Уилсон. Введение в теорию графов/ Р.Уилсон. – М.: Мир, 1977.
13. Д. Кук . Компьютерная математика/ Д. Кук, Г. Бейз. – М.: Наука, 1990.
14. Лекции по теории графов/ Емеличев В.А. и др. – М.: Наука, 1990.
15. О. Оре. Теория графов/ О.Оре. – М.: Наука, 1980.
16. С.П. Гаврилов Сборник задач по дискретной математике/ С.П. Гаврилов, А.А. Сапоженко. – М.: Наука, 1978.
17. Лавров И.А. Задачи по теории множеств, математической логике и теории алгоритмов./ И.А. Лавров, Л.Л Максимов. – М.: ФИЗМАТЛИТ, 2001.
18. В.И. Игошин. Задачи и упражнения по математической логике и теории алгоритмов/ В.И. Игошин. – М.: Издательский центр "Академия", 2007.
19. Карпов Ю.Г. Теория автоматов/ Ю.Г. Карпов. – СПб.: Питер, 2002.
20. Гавриленко С.Ю. Основи побудови компіляторів: навч. посіб./ С.Ю. Гавриленко, М.І.Главчев, А.М. Філоненко. – Харків: Курсор, 2005. – 107 с
21. Системне програмування. Системні сервісні компоненти /Дерев'янко О.С., Межерицький С.Г., Гавриленко С.Ю., Клименко А.Н. – Харків: НТУ "ХПІ", 2009. – 160 с.
22. И.А. Волкова. Формальные языки и грамматики. Элементы теории трансляции/ И.А. Волкова, Т.В. Руденко. – М.:МГУ, 1999. – 62 с.
23. А.В. Бржезовский Лексический и синтаксический анализ. Формальные языки и грамматики. – Л.: ЛИАП, 1990. – 62 с.
24. Альфред Ахо. Компиляторы. Принципы, технологии, инструменты/ Альфред Ахо, Равви Сети, Джеффри Ульман.– М., СПБ., К., 2001. – 768 с.

ЗМІСТ

Навчальне видання

ГАВРИЛЕНКО Світлана Юріївна
КЛИМЕНКО Алла Миколаївна
ЛЮБЧЕНКО Наталія Юріївна
СМОЛЯР Віктор Григорович
ТИШКО Сергій Олександрович

**ТЕОРІЯ ЦИФРОВИХ АВТОМАТІВ
ТА ФОРМАЛЬНИХ МОВ.
ВСТУПНИЙ КУРС**

Навчальний посібник
для комп'ютерних спеціальностей вищих навчальних закладів
напрямів 6.050102 " Комп'ютерна інженерія" та
6.050101 "Комп'ютерні науки"

Роботу до видання рекомендував *М.Й. Заполовський*

Редактори *Л.А. Пустовойтова, О.С. Самініна*

План 2011 р., поз. 20/
Підписано до друку . .10. Формат 60x84 1/16. Папір офсет. №2.
Друк – ризографія. Гарнітура Times New Roman. Ум. друк. арк. 10,0
Обл.-вид. арк. 12,1 Наклад 300 прим. Зам №. Ціна договірна.
Видавничий центр НТУ "ХПІ", 61002, Харків, вул. Фрунзе, 21.
Свідоцтво про державну реєстрацію ДК № 116 від 10.07.2000 р.

Друкарня НТУ "ХПІ", 61002, Харків, вул. Фрунзе, 21