Міністерство освіти і науки, молоді та спорту України

Прикарпатський національний університет
імені Василя Стефаника

Володимир Гаврилків

Формальні мови та алгоритмічні моделі

Навчальний посібник
для студентів напрямів підготовки
«математика», «прикладна математика»
вищих навчальних закладів

Івано-Франківськ

2012

УДК 510.5:004.423.24
ББК 22.123
Г 12

Гаврилків В.М. Формальні мови та алгоритмічні моделі: навчальний посібник / В.М. Гаврилків. — Івано-Франківськ: «Сімик», 2012. — 172 с.

В посібнику у вигляді курсу з 18 параграфів викладено основи теорії формальних мов і скінченних автоматів та розглянуто три основні алгоритмічні моделі: машина Тюрінґа, рекурсивні функції та нормальні алгоритми Маркова. Кожен параграф супроводжується питаннями та завданнями для самостійного розв'язування.

Рекомендовано Вченою радою факультету математики та інформатики як навчальний посібник для студентів напрямів підготовки «математика», «прикладна математика» (протокол № 1 від 6 вересня 2012 р.).

Рецензенти: доктор фізико-математичних наук, професор
Протасов Ігор Володимирович, провідний науковий співробітник кафедри дослідження операцій Київського національного університету імені Тараса Шевченка;
доктор фізико-математичних наук, професор
Банах Тарас Онуфрійович, професор кафедри геометрії і топології Львівського національного університету імені Івана Франка;
кандидат фізико-математичних наук
Бандура Андрій Іванович, доцент кафедри вищої математики Івано-Франківського національного технічного університету нафти і газу.

Зміст

Передмова

За останні тридцять років ми стали свідками дуже бурхливого зростання математичних досліджень, пов'язаних з комп'ютерами та обчислювальною технікою. Величезний розвиток обчислювальної техніки відкрив нові горизонти для розвитку математики, який за швидкістю не має аналогів у довгій історії математики. З одного боку, нові дослідження в математиці безпосередньо ініціювали розвиток комп'ютерів і комп'ютерних наук. З іншого боку, досягнення в галузі комп'ютерних наук сприяли енергійному розвитку деяких розділів математики. Це, зокрема, стосується дискретної математики і таких її розділів як теорія формальних мов і скінченних автоматів та теорія алгоритмів. Ці області складають захоплюючу частину сучасної математики, і їм притаманний дуже динамічний характер розвитку, який повний складних завдань, що вимагають цікавих і винахідливих математичних методів. Даний посібник є гарною основою для розуміння цих областей математики.

Загальновизнано, що найкращий спосіб вивчення математики ґрунтується на розв'язуванні прикладів та практичних задач. Практичний підхід подачі матеріалу не тільки допомагає вивчити методи та інструменти для розв'язання задач, а й зміцнює розуміння основних понять. Книзі притаманний послідовний виклад матеріалу з великою кількістю ілюстративних прикладів. Вона містить понад 200 вправ, які пояснюють і розширюють матеріал.

Посібник має вигляд курсу з 18 параграфів, поділених на три розділи. У першому розділі розглядаються формальні мови (зокрема, регулярні та контекстно-вільні мови), системи числення та граматики. Другий розділ містить виклад основних понять і фактів, пов'язаних зі скінченними автоматами. Матеріал третього розділу містить детальний розгляд трьох основних алгоритмічних моделей: машин Тюрінґа, рекурсивних функцій та нормальних алгоритмів Маркова. Останній параграф присвячений питанням складності алгоритмів. Кожен параграф супроводжується питаннями для (само)контролю засвоєння та вправами, які є основою для проведення практичного заняття з даної теми.

Розділ I

Формальні мови

§ 1. Вільні напівгрупи і формальні мови

В цьому параграфі вводиться поняття формальної мови та вивчаються деякі властивості формальних мов. Вивчення формальних мов спричинено, в першу чергу, їх застосуванням в ком'ютерних науках. Формальні мови використовуються для опису штучних мов, наприклад, мов програмування, мов команд операційної системи. Будь-яке програмне забезпечення створюється і експлуатується з допомогою тієї чи іншої формальної мови.

Розпочнемо параграф з необхідних допоміжних означень. Нехай X – довільна множина. *Бінарною операцією на множині X* називається відображення $* : X \times X \to X$, де $X \times X$ – множина всіх впорядкованих пар елементів з X. Образ в X елемента $(a, b) \in X \times X$ позначатимемо через $a * b$ або просто через ab. Непорожня множина X, наділена бінарною операцією $* : X \times X \to X$, називається *групоїдом*. Бінарна операція $*$ на множині X називається *асоціативною*, якщо $a(bc) = (ab)c$ для всіх $a, b, c \in X$. Якщо операція $*$ є асоціативною на X, то пара $(X, *)$ називається *напівгрупою*. Нехай $(X, *)$ і $(Y, \circ)$ – напівгрупи. *Гомоморфізмом з X в Y* називається таке відображення $\varphi : X \to Y$, що $\varphi(a * b) = \varphi(a) \circ \varphi(b)$ для всіх $a, b \in X$.

Нехай X – непорожня множина, яку називатимемо *(формальним) алфавітом*, а її елементи – *буквами (символами, знаками)*. Якщо $|X| = 2$, то алфавіт називається *бінарним, або двійковим*. Найчастіше бінарний алфавіт позначається через $B = \{0, 1\}$.

1.1. Приклад. Алфавіт X формул алгебри висловлень є об'єднанням $X = X_1 \cup X_2 \cup X_3$, де множина $X_1 = \{p, q, r, \ldots\}$ містить пропозиційні змінні, $X_2 = \{\wedge, \vee, \to, \leftrightarrow, \neg\}$ – логічні зв'язки, а множина $X_3 = \{(,)\}$ складається з допоміжних символів (дужок).

Визначимо *слово (ланцюг)* в алфавіті X як непорожню скінченну послідовність $x_1x_2\dots x_m$ елементів з X. Наприклад, 010111 – слово в бінарному алфавіті B, а $p\wedge q\to r$ – слово в алфавіті X з прикладу 1.1. Таким чином, два слова $x_1x_2\dots x_m$ і $y_1y_2\dots y_n$ дорівнюють тоді і тільки тоді, коли вони співпадають як послідовності, тобто коли $m=n$ і $x_1=y_1,\dots,x_m=y_m$.

Алфавіти позначатимемо великими буквами латинського алфавіту, наприклад A, B, C, X і т.д. Букви формального алфавіту будемо позначати малими буквами латинського алфавіту: a, b, c і т.д., а слова – малими буквами грецького алфавіту, наприклад, α, β, ω.

Через X^+ позначимо множину всіх слів в алфавіті X. Наприклад, якщо X – бінарний алфавіт $\{0,1\}$, то

$$X^+=\{0,1,00,01,10,11,000,001,\dots\}.$$

На множині X^+ визначимо бінарну операцію:

$$x_1x_2\dots x_m\circ y_1y_2\dots y_n=x_1x_2\dots x_my_1y_2\dots y_n.$$

Операція $\circ$ називається *конкатенацією*. Вона, очевидно, асоціативна, і $(X^+,\circ)$ називається *вільною напівгрупою на множині X*. Вільну напівгрупу на множині X позначають часто також через $F(X)$ (від англійського слова „free” – вільний).

Поряд з вільною напівгрупою X^+ над алфавітом X часто розглядають вільний моноїд $X^*=X^+\cup\{e\}$ над X, в якому одиниця e є порожнім словом, яке не містить жодної букви.

Слово, яке містить i букв (слів) a позначатимемо через a^i. Наприклад, $a^1=a$ (таким чином, ми ототожнюємо букву a з словом, що містить єдину букву a), $a^2=aa$, $a^0=e$ – порожнє слово e. *Оберненням слова x* (позначається через x^R) називається слово, записане в оберненому порядку, тобто якщо $x=a_1\dots a_n$, де всі a_i – букви, то $x^R=a_n\dots a_1$. Крім того, $e^R=e$. Легко перевірити, що $(xy)^R=y^Rx^R$.

Нехай α, β і γ – довільні слова в деякому алфавіті X. Назвемо α *префіксом слова $\alpha\beta$*, а β *– суфіксом слова $\alpha\beta$*. Слово β називатимемо *підсловом слова $\alpha\beta\gamma$*. Префікс і суфікс слова є його підсловами. Наприклад, ba – префікс і підслово слова bac. Зауважимо, що порожнє слово є префіксом, суфіксом і підсловом довільного слова.

Якщо $\alpha\neq\beta$ і α – префікс (суфікс) слова β, то α називається *власним префіксом (суфіксом) слова β*. *Довжина слова* – це

кількість букв у ньому, тобто якщо $\omega = a_1 \dots a_n$, де всі a_i – букви, то довжина слова ω дорівнює n. Довжину слова ω позначатимемо через $|\omega|$. Наприклад, $|aab| = 3$ і $|e| = 0$. Очевидно, що $|\omega_1 \circ \omega_2| = |\omega_1| + |\omega_2|$.

Формальною мовою в алфавіті X називається довільна множина слів в X, тобто довільна підмножина вільного моноїда X^*. Наприклад, множина $\emptyset$ – це формальна мова. Множина $\{e\}$, яка містить тільки порожнє слово, також є мовою. Зауважимо, що $\emptyset$ і $\{e\}$ – дві різні формальні мови. Іншим прикладом формальної мови є мова L, що містить всі слова, які складаються з нуль і більше букв a. Її можна позначити через $\{a^i : i \geq 0\}$. Ясно, що $L = \{a\}^*$ (для спрощення $\{a\}^*$ позначатимемо надалі через a^*).

Якщо мова L така, що жодне слово в L не є власним префіксом (суфіксом) жодного іншого слова в L, то кажуть, що L має *префіксну (суфіксну) властивість*. Наприклад, a^* не має префіксної властивості, а $\{a^i b : i \geq 0\}$ має.

Оскільки формальна мова L – це множина (підмножина X^*), то до формальних мов застосовні теоретико-множинні операції об'єднання, перетину, знаходження різниці і доповнення. Операцію конкатенації можна застосувати до мов таким же чином, як і до слів.

Нехай L_1 та L_2 – формальні мови в алфавітах X_1 та X_2 відповідно. *Конкатенацією (добутком) мов L_1 і L_2* називається формальна мова $L_1 L_2 = \{\omega_1 \circ \omega_2 : \omega_1 \in L_1, \omega_2 \in L_2\}$ в алфавіті $X_1 \cup X_2$, яка складається зі слів, які утворюються шляхом дописування до будь-якого слова мови L_1 довільного слова мови L_2.

Ітерація (замикання Кліні) мови L, яка позначається через L^*, визначається наступним чином:

1) $L^0 = \{e\}$,
2) $L^n = LL^{n-1}$ для $n \geq 1$,
3) $L^* = \cup_{n \geq 0} L^n$.

Розглянемо деякі приклади.

1.2. Приклад. Знайдемо ітерацію мови $L = \{00, 111\}$. Мова $L^0 = \{e\}$ містить одне порожнє слово незалежно від самої мови L. $L^1 = L$ містить всі слова мови L. Таким чином, перші два члени в розкладі L^* утворюють мову $\{e, 00, 111\}$. Далі знайдемо мову $L^2 = LL = \{0000, 00111, 11100, 111111\}$. Аналогічно $L^3 = LL^2$ є множиною слів,

утворених трикратним вибором із двох слів мови L. Для обчислення L^* необхідно знайти L^i для кожного $i \in \mathbb{N} \cup \{0\}$ і об'єднати всі ці мови. Хоча кожна множина L^i є скінченною, об'єднання нескінченної кількості множин, як правило, утворює нескінченну мову. Це, зокрема, стосується і мови L, яка містить нескінченну кількість слів, в яких максимальні підслова лише з нулів мають парну довжину, а довжина максимальних слів з одиниць кратна три.

1.3. Приклад. Покажемо, що формальна мова $\{0, 10\}^*$ складається з усіх слів, які не містять підслова 11 і закінчуються на 0.

Очевидно, що конкатенація довільної кількості 0 та 10 закінчується на 0. Вона не може породити підслово 11, оскільки останні букви обох слів 0 та 10 розділяють довільні дві 1 в їхній конкатенації.

Нехай ω – слово в алфавіті $\{0, 1\}$, яке не містить підслів 11 і закінчується на 0. Якщо ω не містить входжень 1, то ω є конкатенацією $|\omega|$ букв 0, і тому $\omega \in \{0, 10\}^{|\omega|} \subset \{0, 10\}^*$. Нехай слово ω містить $n \geq 1$ входжень букв 1. Тоді після кожного входження букви 1 в ω мусить слідувати буква 0, бо інакше буква 1 або слідує за 1, або є останньою буквою слова ω, що суперечить вибору слова ω. Таким чином, слово ω запишеться як

$$0 \dots 0(10)0 \dots 0(10)0 \dots 0(10)0 \dots 0,$$

де вираз $0 \dots 0$ означає нуль або більше входжень букви 0. Отже, ω є конкатенацією слів 0 та 10, тобто $\omega \in \{0, 10\}^*$.

Для мови L визначимо *обернену мову* як $L^R = \{\omega^R : \omega \in L\}$.

1.4. Приклад. Доведемо, що для формальних мов A і B виконуються рівності $(AB)^R = B^R A^R$ та $(A \cup B)^R = A^R \cup B^R$.

Дійсно, $(AB)^R = \{\omega^R \mid \omega \in AB\} = \{(\alpha\beta)^R \mid \alpha \in A,\ \beta \in B\} = \{\beta^R \alpha^R \mid \alpha \in A,\ \beta \in B\} = \{\beta^R \mid \beta \in B\} \cdot \{\alpha^R \mid \alpha \in A\} = B^R A^R$.

Аналогічно $(A \cup B)^R = \{\omega^R \mid \omega \in A \cup B\} = \{\omega^R \mid \omega \in A\} \cup \{\omega^R \mid \omega \in B\} = A^R \cup B^R$.

1.5. Твердження. *(Лема Ардена) Нехай A і B – дві формальні мови, причому $e \notin A$, і мова X задовольняє рівність $X = AX \cup B$. Тоді $X = A^*B$.*

Доведення. Доведемо індукцією за довжиною $|\omega|$ слова ω, що $X \subset A^*B$. Спершу розглянемо випадок $|\omega| = 0$, тобто $\omega = e$. Якщо $e \in X$, то $e \in AX \cup B$. Оскільки $e \notin A$, то необхідно $e \in B$ і, отже, $e \in A^*B$.

Далі припустимо, що для всіх слів ω довжини менше n з того, що $\omega \in X$ випливає $\omega \in A^*B$, і розглянемо слово α довжини n. Якщо $\alpha \in X = AX \cup B$, то або $\alpha \in B \subset A^*B$, або $\alpha = \beta\gamma$ для деяких $\beta \in A$ і $\gamma \in X$. В другому випадку $\beta \neq e$ і, отже, $|\gamma| < |\alpha|$. Тому за індуктивним припущенням $\gamma \in A^*B$ і $\alpha = \beta\gamma \in AA^*B \subset A^*B$. Цим завершується індуктивний крок і, отже, $X \subset A^*B$.

Щоб довести протилежне включення, використаємо індукцію, щоб показати, що $A^nB \subset X$ для всіх $n \geq 0$. Для $n = 0$ маємо, що $A^0B = B \subset AX \cup B = X$. Для $n > 0$ за індуктивним припущенням виконується $A^nB = A(A^{n-1}B) \subset AX$. Таким чином, $A^nB \subset AX \subset AX \cup B = X$ і $A^*B = (\cup_{n\geq 0}A^n)B = \cup_{n\geq 0}(A^nB) \subset X$. □

1.6. Приклад. Нехай формальні мови $A, B \subset \{a, b\}^*$ задовольняють рівності:

$$A = \{e\} \cup \{a\}A \cup \{b\}B,$$
$$B = \{e\} \cup \{b\}B.$$

Знайти мови A і B.

Застосуємо лему Ардена до другого рівняння і знайдемо мову $B = \{b\}^*\{e\} = \{b\}^*$. Аналогічно знаходимо мову $A = \{a\}^*(\{e\} \cup \{b\}B)$.

Насамкінець підставимо $\{b\}^*$ замість B і одержуємо:

$$A = \{a\}^*(\{e\} \cup \{b\}\{b\}^*) = \{a\}^*\{b\}^*.$$

Нехай X_1 і X_2 – алфавіти і $h : X_1 \to X_2^*$ – довільне відображення. Відображення h можна продовжити до гомоморфізму $\overline{h} : X_1^* \to X_2^*$, покладаючи $\overline{h}(e) = e$ і $\overline{h}(\omega a) = \overline{h}(\omega)h(a)$ для всіх $\omega \in X_1^*$ і $a \in X_1$. Застосовуючи гомоморфізм до мови L, одержимо іншу мову $\overline{h}(L)$, яка є множиною слів $\{\overline{h}(\omega) : \omega \in L\}$.

1.7. Приклад. Припустимо, що потрібно замінити кожне входження в слово букви 0 на букву a, а кожне входження 1 на bb. Тоді можна визначити гомоморфізм h так, що $h(0) = a$ і $h(1) = bb$. Якщо $L = \{0^n 1^n : n \geq 1\}$, то $\overline{h}(L) = \{a^n b^{2n} : n \geq 1\}$.

Рекомендована література : [**2**, с. 26–31], [**4**, с. 460–471], [**5**, с. 338–341], [**23**, с. 6–14].

Питання та вправи до параграфа 1.

1.1. Дайте означення формального алфавіту та формальної мови.

1.2. Де застосовуються формальні мови?

1.3. Які операції над формальними мовами ви знаєте?

1.4. Визначте формальні алфавіти для запису:

а) арифметичних виразів;
б) азбуки Морзе.

1.5. Визначте довжину слова $\alpha =$ „математика” в українському алфавіті.

1.6. Які з наведених слів є підсловами слова $\omega =$ „індустралізація”: $\alpha_1 =$ „індус”, $\alpha_2 =$ „індустрія”, $\alpha_3 =$ „ліза”, $\alpha_4 =$ „акція”?

1.7. Випишіть всі (власні) префікси, суфікси і підслова слова $abca$.

1.8. Скільки формальних мов можна побудувати на алфавіті $X = \{0, 1, 2, 3\}$, в яких довжина кожного слова не перевищує 3?

1.9. Скільки слів ω довжини $|\omega| \leq n$ можна побудувати в алфавіті, що містить k букв? Скільки формальних мов можна утворити з цих слів?

1.10. Знайдіть всі слова ω в алфавіті $\{0, 1\}$, які задовольняють рівність $\omega 011 = 011\omega$.

1.11. Чи правда, що якщо формальна мова A містить n слів, а мова B – m слів, то AB мусить містити nm слів?

1.12. Нехай $A = \{(01)^n : n \geq 0\}$ і $B = \{01, 010\}$. Знайти $A \cup B$, $A \cap B$, $A \setminus B$, $B \setminus A$, $A \Delta B$, AB і ABA.

1.13. Нехай $A = \{\text{пра}, e\}$, $B = \{\text{дід, баба}\}$. Знайти AB і A^*B.

1.14. Чи може мова L^* або L^+ бути порожньою? За яких умов мови L^* і L^+ є скінченними?

1.15. Знайти слово найменшої довжини в алфавіті $\{0\}$, яке не належить формальній мові $\{e, 0, 0^2, 0^5\}^3$.

1.16. Довести рівність: $(A \cup B)^* = A^*(BA^*)^*$.

1.17. Нехай A і B – формальні мови. Довести або спростувати рівності:

1) $(A^R)^* = (A^*)^R$;
2) $(A^+)^* = A^*$;
3) $(A \cup A^R)^* = A^* \cup (A^*)^R$;
4) $A^2 \cup B^2 = (A \cup B)^2$;
5) $A^* \cap B^* = (A \cap B)^*$.

1.18. Які з наступних мов мають префіксну (суфіксну) властивість?

1) $\emptyset$;
2) $\{e\}$;
3) $\{a^n b^n : n \geq 1\}$;
4) L^*, якщо L має префіксну властивість;
5) $\{\omega : \omega \in \{a, b\}^*$ і кількість букв a в ω дорівнює кількості букв $b\}$.

1.19. Нехай h – гомоморфізм, визначений рівностями $h(0) = a$, $h(1) = bb$ і $h(2) = e$. Опишіть формальну мову $\overline{h}(L)$, де $L = \{012\}^*$.

§ 2. Системи числення

В цифрових обчислювальних машинах будь-яка інформація, задана певною формальною мовою, зображується в вигляді строго визначеної послідовності цифр. Кожній такій послідовності можна поставити в відповідність певне число і звести, таким чином, будь-яке перетворення інформації до операцій над числами, заданих у деякій системі числення. В зв'язку з цим у даному параграфі ми детально зупинимося на різних числових системах.

Система числення (англ. number (numeration) system, notation) – це сукупність способів і засобів запису чисел для проведення підрахунків.

Найпростішою системою числення є *унарна система числення*, в якій кожне натуральне число зображується відповідною кількістю деяких символів. Наприклад, якщо вибрати символ „|", то натуральне число сім запишеться у вигляді |||||||. Унарна система числення використовується для запису малих чисел і має застосування в теорії алгоритмів.

2.1. Типи систем числення. Розрізняють такі типи систем числення: *позиційні, непозиційні* та *змішані*. Найбільш поширеними є позиційні системи числення. У них одна і та ж цифра (числовий знак) у записі числа набуває різних значень залежно від своєї позиції. Кожна позиція з присвоєним їй порядковим номером називається *розрядом числа*. Домовимося про наступну нумерацію розрядів: якщо число має n розрядів цілої і m розрядів дробової частини, то старшому розряду цілої частини присвоюється номер $n-1$, молодшому розряду цілої частини – номер 0, старшому розряду дробової частини – номер -1, а молодшому розряду – номер -m. В позиційних системах числення кожному розряду присвоюється відповідна *вага*. Найчастіше використовуються позиційні системи числення, в яких i-му розряду присвоюється вага g^i, де $g \in \mathbb{N} \setminus \{1\}$. Натуральне число g при цьому називають *основою системи числення*. Значення числа за його зображенням визначається за формулою:

$$(1) \qquad (a_{n-1}a_{n-2}\dots a_i\dots a_1a_0, a_{-1}a_{-2}\dots a_{-m})_g = \sum_{i=-m}^{n-1} a_i g^i =$$

$$a_{n-1}g^{n-1}+a_{n-2}g^{n-2}+\dots+a_ig^i+\dots+a_1g+a_0g^0+a_{-1}g^{-1}+\dots+a_{-m}g^{-m}$$

В лівій частині формули (1) записано символічне зображення числа. Тут a_i $(i = n-1, n-2, \dots, -m)$ – символи, які позначають деякі цілі числа. Права частина формули (1) визначає правило обчислення числового значення символічного запису лівої частини цієї формули. Можна вважати, що ліва частина формули (1) є шифром

числа, а права частина визначає алгоритм дешифрування даного шифру.

2.1. Приклад. Обчислимо значення числа $10(-1)(-1),01$ в системі числення з основою 3 і символами -1, 0 та 1. В цьому випадку кількість цілих розрядів $n = 4$, а кількість дробових розрядів $m = 2$. З допомогою формули (1) одержуємо:

$$10(-1)(-1),01 = 1\cdot 3^3 + 0\cdot 3^2 + (-1)\cdot 3 + (-1) + 0\cdot 3^{-1} + 1\cdot 3^{-2} = 23\frac{1}{9}.$$

Найчастіше (хоча і не завжди) символи a_i позначають цілі числа від 0 до $g-1$ і називаються *цифрами* даної системи числення, а сама система числення називається *g-ковою системою числення*. В g-ковій системі числення кількість різних цифр дорівнює основі g системи числення. Наприклад, в звичайній арабській (десятковій) системі числення використовуються цифри $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$.

У *непозиційних системах числення* величина, яку позначає цифра, не залежить від її позиції у числі. При цьому система може накладати обмеження на позиції цифр, наприклад, щоб вони були розташовані по спаданню чи згруповані за значенням. Проте це не є принциповою умовою для розуміння записаних такими системами чисел.

Типовим прикладом непозиційної системи числення є *римська система числення*. У ній в якості цифр використовуються латинські букви:

Римська цифра	Десяткове значення
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

Натуральні числа записуються за допомогою повторення цих цифр. При цьому, якщо більша цифра стоїть перед меншою, то вони додаються (принцип додавання), якщо ж менша — перед більшою, то менша віднімається від більшої (принцип віднімання). Останнє

правило застосовується тільки для уникнення чотириразового повторення однієї цифри. Римські цифри I, X, C ставляться відповідно перед X, C, M для позначення 9, 90, 900 або перед V, L, D для позначення 4, 40, 400. Наприклад, $VI = 5 + 1 = 6$, $IV = 5 - 1 = 4$, $XIX = 10 + 10 - 1 = 19$ (замість $XVIIII$), $XL = 50 - 10 = 40$, $XXXIII = 10 + 10 + 10 + 1 + 1 + 1 = 33$ тощо.

Римські числа використовувалися стародавніми римлянами. Виконання арифметичних дій над багатозначними числами в цій системі дуже незручне. Дана система числення на сьогодні майже не застосовується. З допомогою таких чисел позначають століття (XV ст.), роки н.е. ($MCMLXXVII$) та місяці в датах (1.V.1975), порядкові числівники, а також похідні невеликих порядків (y^{IV}, y^{V}). Часто римські числа використовують також із естетичною метою.

Змішана система числення є узагальненням системи числення з основою g і її часто відносять до позиційних систем числення. Основою змішаної системи є послідовність чисел, що зростає, $\{b_k\}_{k=0}^{n}$ і кожне число x записується як лінійна комбінація: $x = \sum_{k=0}^{n} a_k b_k$, де на коефіцієнти a_k (цифри) накладаються деякі обмеження. Якщо $b_k = g^k$ для деякого g, а $a_k \in \{0, 1, \ldots, g-1\}$, то змішана система співпадає з g-ковою системою числення.

Найвідомішим прикладом змішаної системи числення є зображення часу у вигляді кількості діб, годин, хвилин і секунд. При цьому величина d днів h годин m хвилин s секунд відповідає значенню $d \cdot 24 \cdot 60 \cdot 60 + h \cdot 60 \cdot 60 + m \cdot 60 + s$ секунд.

Цікавим прикладом є *система числення майя*. Майя використовували двадцяткову систему числення за одним винятком: у другому розряді було не 20, а 18 ступенів, тобто після числа (17)(19) відразу йшло число (1)(0)(0). Це було зроблено для полегшення розрахунків календарного циклу, оскільки (1)(0)(0) дорівнювало 360, що приблизно дорівнює кількості днів у сонячному році.

У нумізматиці особливо велику вагу мають десяткова, дванадцяткова (дуодецимальна), четвіркова та шісткова системи числення. У інформаційних технологіях застосовуються двійкова, десяткова, вісімкова та шістнадцяткова системи числення.

2.2. Позиційні g-кові системи числення. Зупинимося детальніше на g-кових позиційних системах числення. Спершу доведемо наступну лему.

2.2. Лема. *Якщо $a, g \in \mathbb{Z}$ і $g > 0$, то існують такі єдині числа $s, r \in \mathbb{Z}$, що $a = sg + r$, де $0 \le r < g$.*

Доведення. Розглянемо множину всіх цілих чисел виду $a - xg$, де $x \in \mathbb{Z}$. Нехай $r = a - sg$ – найменший невід'ємний елемент цієї множини. Ми стверджуємо, що $0 \le r < g$. Дійсно, в протилежному випадку $a - sg \ge g$, а тому $0 \le a - (s+1)g < r$, що суперечить мінімальності r. Доведемо єдиність даного зображення. Припустимо, що існує інший запис $a = \tilde{s}g + \tilde{r}$, де $0 \le \tilde{r} < g$. Тоді $a = sg + r = \tilde{s}g + \tilde{r}$, звідки $(s - \tilde{s})g = \tilde{r} - r$ і $-g < \tilde{r} - r < g$. Оскільки $\tilde{r} - r$ ділиться на g, то $\tilde{r} - r = 0$, а отже, $\tilde{r} = r$ і $\tilde{s} = s$. □

Нехай $g \in \mathbb{N} \setminus \{1\}$ – основа g-кової системи числення.

2.3. Теорема. *Кожне натуральне число m можна єдиним чином подати у вигляді*

$$m = a_n g^n + a_{n-1} g^{n-1} + \ldots + a_1 g + a_0,$$

де a_i $(0 \le i \le n)$ – деякі цілі невід'ємні числа, менші від g, причому $a_n \ne 0$.

Доведення. Згідно з лемою 2.2 для натуральних чисел m та g знайдуться такі єдині цілі числа s_0 і a_0, що $m = s_0 g + a_0$ і $0 \le a_0 < g$. Оскільки $g \ge 2$, то $0 \le s_0 < m$. Якщо $s_0 = 0$, то теорема доведена, інакше існують такі єдині цілі числа s_1 і a_1, що $s_0 = s_1 g + a_1$, $0 \le a_1 < g$ і $0 \le s_1 < s_0$. Продовжуючи даний процес аналогічно, одержимо строго спадну послідовність (s_k) натуральних чисел. Дана послідовність не може бути нескінченною, а тому існує таке n, що $s_{n-1} \ne 0$, але $s_n = 0$. Звідси одержуємо, що $s_{n-2} = s_{n-1}g + a_{n-1}$ і $s_{n-1} = s_n g + a_n = a_n$, де $0 \le a_i < g$. Тоді шуканий єдиний запис числа m матиме вигляд $m = s_0 g + a_0 = (s_1 g + a_1)g + a_0 = s_1 g^2 + a_1 g + a_0 = \ldots = s_{n-2} g^{n-1} + a_{n-2} g^{n-2} + \ldots + a_1 g + a_0 = (s_{n-1} g + a_{n-1}) g^{n-1} + a_{n-2} g^{n-2} + \ldots + a_1 g + a_0 = a_n g^n + a_{n-1} g^{n-1} + a_{n-2} g^{n-2} + \ldots + a_1 g + a_0$. □

Можна довести, що для кожного додатнього дробового числа a так само, як і для кожного натурального числа, в системі числення з будь-якою основою g існує тільки єдиний запис виду

$$a = \sum_{i=-m}^{n} a_i g^i = a_n g^n + \ldots + a_i g^i + \ldots + a_0 g^0 + a_{-1} g^{-1} + \ldots + a_{-m} g^{-m}.$$

Отже, *запис числа a в системі числення з основою g* – це зображення числа у вигляді суми степенів основи g з цілими невід'ємними коефіцієнтами, меншими ніж основа g.

Вираз

$$a = a_n g^n + \ldots + a_i g^i + \ldots + a_0 g^0 + a_{-1} g^{-1} + \ldots + a_{-m} g^{-m}.$$

скорочено записують так

$$a = (a_n a_{n-1} \ldots a_i \ldots a_1 a_0, a_{-1} a_{-2} \ldots a_{-m})_g.$$

Введення знака мінус дає змогу записувати у системі числення з основою g також і від'ємні числа.

2.3. Арифметичні операції в різних системах числення. При виконанні арифметичних операцій над числами, записаними в десятковій системі числення, ми користуємось правилами додавання, віднімання і множення чисел „стовпцем" і ділення „кутом". За цими ж правилами виконують операції й над числами, записаними в будь-якій іншій системі числення. Доведемо, наприклад, правило додавання в системі числення з основою g. Нехай $a = (a_k \ldots a_1 a_0)_g$ і $b = (b_s \ldots b_1 b_0)_g$. Не втрачаючи загальності, можна вважати, що $k \geq s$. Знайдемо суму $a + b$.

$a+b = (a_k \ldots a_1 a_0)_g + (b_s \ldots b_1 b_0)_g = (a_k g^k + \ldots + a_{s+1} g^{s+1} + a_s g^s + \ldots + a_1 g + a_0) + (b_s g^s + \ldots + b_1 g + b_0) = a_k g^k + \ldots + a_{s+1} g^{s+1} + (a_s + b_s) g^s + \ldots + (a_1 + b_1) g + (a_0 + b_o)$.

У цьому записі деякі з чисел $a_0 + b_0, a_1 + b_1, \ldots, a_s + b_s$ можуть виявитися більшими або дорівнювати основі числення g. Якщо $a_m + b_m \geq g$ $(0 \leq m \leq s)$, то $a_m + b_m = g + r_m$, де $0 \leq r_m < g$. Тому $(a_{m+1} + b_{m+1}) g^{m+1} + (a_m + b_m) g^m = (a_{m+1} + b_{m+1} + 1) g^{m+1} + r_m g^m$. Замінивши кожен вираз $(a_m + b_m) \geq g$, починаючи з $a_0 + b_0$ і закінчуючи $a_s + b_s$, рівнозначним йому виразом $g + r_m$ і перенісши там, де це потрібно, одиницю в наступний розряд, дістанемо запис суми $a + b$ в системі числення з основою g:

$$a + b = c_t g^t + \ldots + c_1 g + c_o,$$

або скорочено $a + b = (c_t \ldots c_1 c_o)_g$.

Таким чином, щоб додати два цілі додатні числа, записані в системі числення з основою g, потрібно додати їхні цифри першого розряду, потім цифри другого розряду і т.д. При цьому кожного разу, коли при додаванні цифр даного розряду дістанемо суму більшу або рівну основі системи числення g, потрібно перенести одиницю в наступний розряд. При додаванні чисел доданки доцільно підписувати один під одним (у стовпчик) так, щоб цифри, які відповідають однаковим розрядам, були одна під одною. Як бачимо, додавання багатоцифрових чисел зводиться до додавання одноцифрових чисел. Таким чином, основою додавання системних чисел є таблиця додавання одноцифрових чисел, що визначає суму двох чисел, менших, ніж основа g.

Можна показати, що віднімання, множення і ділення чисел виконується цілком аналогічно правилам звичайної шкільної десяткової арифметики.

Побудуємо, наприклад, таблиці додавання і множення одноцифрових чисел в четвірковій системі числення:

+	**0**	**1**	**2**	**3**
0	0	1	2	3
1	1	2	3	10
2	2	3	10	11
3	3	10	11	12

·	**0**	**1**	**2**	**3**
0	0	0	0	0
1	0	1	2	3
2	0	2	10	12
3	0	3	12	21

У кожній клітинці цих таблиць записано у четвірковій системі числення суму (добуток) цифр, що є номерами рядка і стовпця, на перетині яких стоїть дана клітинка. Користуючись цими таблицями і правилами, аналогічними правилам десяткової арифметики, виконаємо арифметичні операції над числами в четвірковій системі числення:

$$\begin{array}{r} +203,2_4 \\ 132,1_4 \\ \hline 1001,3_4 \end{array} \qquad \begin{array}{r} -203,2_4 \\ 132,1_4 \\ \hline 11,1_4 \end{array} \qquad \begin{array}{r} 102_4 \\ 21_4 \\ \hline 102_4 \\ +210_4 \\ \hline 2202_4 \end{array}$$

2.4. Переведення цілих чисел з однієї позиційної системи числення в іншу. У процесі розв'язування задач доводиться переводити цілі числа з однієї позиційної системи в іншу. Як же перевести число a, записане в системі числення з основою s, в систему числення з основою g? Як відомо, записати число a в системі числення з основою g – це зобразити його у вигляді суми $a = a_k g^k + \ldots + a_1 g + a_o$. Знайдемо коефіцієнти $a_0, a_1, \ldots, a_k$. Поділимо в системі числення з основою s число a на g, дістанемо $a = b_0 g + a_0$, де $a_0 < g$. Далі поділимо b_0 на g: $b_0 = b_1 g + a_1$, де $a_1 < g$. Звідси

$$a = b_0 g + a_0 = (b_1 g + a_1) g + a_0 = b_1 g^2 + a_1 g + a_0.$$

Потім поділимо b_1 на g і т.д. Цей процес продовжуватимемо доти, доки не отримаємо частку, яка дорівнює нулю. Внаслідок цього матимемо:

$$a = a_k g^k + \ldots + a_1 g + a_o.$$

Оскільки за теоремою 2.3 число a можна подати у такому вигляді єдиним чином, то $a_0, a_1, \ldots, a_k$ є цифрами числа a в системі числення з основою g.

Таким чином, для переведення натурального числа a, заданого в s-ковій системі числення, в g-кову систему числення користуємось наступним правилом: *спочатку записуємо число g в s-ковій системі, а потім виконуємо (в s-ковій системі числення!) кілька ділень числа a_s і послідовно утворюваних часток на число g_s доти, доки не отримаємо частку, яка дорівнює нулю. Здобуті остачі спочатку виражаємо цифрами g-кової системи числення і записуємо в зворотньому порядку. Записані таким чином остачі (це вже цифри в g-ковій системі числення) і є g-ковим записом числа a_s.*

2.4. Приклад. Запишемо число $a = 47_{10}$ в трійковій системі числення:

$$\begin{aligned} 47 &= 15 \cdot 3 + \underline{2} \\ 15 &= 5 \cdot 3 + \underline{0} \\ 5 &= 1 \cdot 3 + \underline{2} \\ 1 &= \underline{0} \cdot 3 + \underline{1} \end{aligned}$$

Звідси випливає, що число a в трійковій системі числення запишеться $a = 1202_3$.

2.5. Приклад. Переведемо число $a = 12210_3$ з трійкової системи числення у вісімкову. Нова основа у трійковій системі числення дорівнює 22. Поділимо послідовно у трійковій системі числення число $a = 12210_3$ на 22_3

$$12210_3 = 201_3 \cdot 22_3 + \underline{11_3}$$
$$201_3 = 2_3 \cdot 22_3 + \underline{10_3}$$
$$2_3 = 0_3 \cdot 22_3 + \underline{2_3}$$

Результати послідовного ділення записані у трійковій системі числення, але ми знаємо, що $11_3 = 4_8$ і $10_3 = 3_8$. Отже, $a = 234_8$.

Перехід від g-кової системи числення до десяткової виконують ще й так. Число a_g записують у вигляді

$$a_g = a_n g^n + \ldots + a_1 g + a_o.$$

Потім замість чисел $a_n, \ldots, a_1, a_0$ і g підставляють їхні десяткові записи і роблять відповідні обчислення. Десятковий запис результату є шуканим числом.

У випадку, коли $g = s^k$ перехід від однієї системи числення до іншої значно спрощується. Дійсно, нехай маємо два записи числа a в g-ковій і s-ковій системах числення:

$$a = (a_n \ldots a_1 a_0)_g = (b_l \ldots b_1 b_0)_s, \text{ тобто}$$
$$a = a_n g^n + \ldots + a_1 g + a_0 = b_l s^l + \ldots + b_1 s + b_0.$$

Покажемо спочатку, що для довільних $c_0, c_1, \ldots, c_{k-1}$, $0 \le c_i < s$,

$$c_{k-1} s^{k-1} + \ldots + c_1 s + c_0 < g. \tag{2}$$

Дійсно,

$$c_{k-1}s^{k-1}+\ldots+c_1 s+c_0 \le (s-1)s^{k-1}+\ldots+(s-1)s+(s-1) = s^k-1 < s^k.$$

З огляду на (2) остача a_0 від ділення числа $a = b_l s^l + \ldots + b_k s^k + b_{k-1} s^{k-1} + \ldots + b_1 s + b_0 = (b_l s^{l-k} + \ldots + b_k) s^k + (b_{k-1} s^{k-1} + \ldots + b_1 s + b_0)$ на $g = s^k$ дорівнює $b_{k-1} s^{k-1} + \ldots + b_1 s + b_0$. Отже, $(a_0)_g = (b_{k-1} \ldots b_1 b_0)_s$. Далі з рівності

$$a_n g^{n-1} + \ldots + a_2 g + a_1 = b_l s^{l-k} + \ldots + b_{k+1} s + b_k.$$

неповних часток від ділення a на $g = s^k$ виливає, що

$$(a_1)_g = (b_{2k-1} \ldots b_{k+1} b_k)_s.$$

Аналогічно можна одержати зображення решти g-кових цифр в вигляді k-розрядних s-кових чисел. Таким чином, для переведення числа, записаного в g-ковій системі числення, в s-кову систему числення достатньо кожну g-кову цифру замінити рівним їй k-розрядним s-ковим числом.

2.6. Приклад. Переведемо число $a = 765,163_8$ в двійкову систему числення:

$$a = 765,163_8 = \underbrace{111}_{7}\underbrace{110}_{6}\underbrace{101}_{5},\underbrace{001}_{1}\underbrace{110}_{6}\underbrace{011}_{3} = 111110101,001110011_2.$$

Для переведення числа з двійкової в вісімкову систему числення достатньо розбити його праворуч і ліворуч від коми на тріади і замінити кожну тріаду відповідною їй вісімковою цифрою. Якщо при розбитті крайні тріади виявляться неповними, то їх слід доповнити нулями.

2.7. Приклад. Переведемо число $a = 1011,1_2$ у вісімкову систему числення:

$$a = 1011,1_2 = \underbrace{001}_{1}\underbrace{011}_{3},\underbrace{100}_{4} = 13,4_8.$$

Розглянемо метод переведення правильних дробів з однієї системи числення в іншу. Нехай правильний дріб a, заданий в s-ковій системі числення, потрібно записати в системі численні з основою g. Припустимо, що запис числа a в g-ковій системі числення знайдений:

$$a = (0, a_{-1}a_{-2}\ldots a_{-m})_g = a_{-1}g^{-1} + a_{-2}g^{-2} + \ldots + a_{-m}g^{-m}.$$

Якщо помножити число a на основу g нової системи числення, то ціла частина добутку буде дорівнювати a_{-1}, а дробова –

$$a_1 = a_{-2}g^{-1} + \ldots + a_{-m}g^{-m+1}.$$

Число a_1 є правильним дробом, оскільки всі цифри $a_i < g$. Таким чином, в результаті множення числа a на g отримуємо цілу частину, яка дорівнює значенню цифри старшого розряду дробового g-кового числа. Продовжуючи множення дробових частин a_i на основу g (цілі частини при кожному множенні відкидаються), можна одержати решту цифр шуканого дробу: ними є цілі частини одержаних добутків. При цьому, якщо $g < s$, то цілі частини добутку є цифрами g-кової системи числення, а якщо $g > s$, то цілі

частини є числами s-кової системи числення, які необхідно замінити цифрами g-кової системи.

Сформулюємо правило переходу від однієї системи числення до іншої методом множення на основу: *щоб перетворити правильний дріб з однієї системи числення в іншу, необхідно послідовно множити це число і проміжні значення на основу нової системи числення (записану в старій системі), відкидаючи кожен раз цілі частини добутків; ці цілі частини є записом цифр шуканого числа g-кової системи числення.*

2.8. Приклад. Переведемо десяткове число $a = 0,6875_{10}$ в двійкову систему числення:

$$\begin{array}{r} \times 0,6875 \\ \hline 2 \\ \hline \times 1,3750 \\ \hline 2 \\ \hline \times 0,7500 \\ \hline 2 \\ \hline \times 1,5000 \\ \hline 2 \\ \hline 1,0000 \end{array}$$

Отже, $a = 0,1011_2$

Слід зауважити, що в загальному випадку дробові числа переводяться з однієї системи числення в іншу наближено, тобто дробова частина добутку при послідовному множенні на основу ніколи не дорівнюватиме нулю. В цьому випадку процес множення продовжують доти, доки не буде одержана необхідна для досягнення заданої точності запису числа кількість розрядів g-кового числа.

2.9. Приклад. Число $\frac{1}{5}$ в десятковій системі числення записують скінченним дробом, а в дванадцятковій – нескінченним:

$$\frac{1}{5} = 0,2_{10} = 0,249724972497\ldots_{12};$$

число $\frac{1}{6}$, навпаки, в дванадцятковій системі числення записують скінченним дробом, а в десятковій – нескінченним:

$$\frac{1}{6} = 0,2_{12}, \frac{1}{6} = 0,1666\ldots_{10}.$$

В обох цих системах число $\frac{1}{4}$ запишеться скінченним дробом, а число $\frac{1}{7}$ – нескінченним:

$$\frac{1}{4} = 0,25_{10} = 0,3_{12}, \frac{1}{7} = 0,142857142857..._{10} = 0,186035186035\ldots_{12}.$$

Для переведення неправильного дробу з однієї системи числення в іншу, слід цілу частину отримати методом ділення, а дробову – методом множення на основу нової системи числення.

Рекомендована література : [**3**, с. 58–80], [**8**, с. 4–11], [**9**, с. 79–88], [**18**, с. 70–89].

Питання та вправи до параграфа 2.

2.1. На які типи поділяються системи числення?

2.2. Які цифри використовуються для запису чисел в римській системі числення?

2.3. Опишіть процес переведення чисел з двійкової системи числення в четвіркову і навпаки.

2.4. Записати в десятковій системі числення такі числа римської нумерації: а) $CXVI$; б) $CXIV$; в) $DXCVI$; г) $MCCII$; д) $MCDXXIX$; е) $MDCCCLXXIV$.

2.5. Записати в римській системі числення числа: а) 39; б) 93; в) 2011; г) 1999.

2.6. Обчислити:

- $(3333_4 + 2222_4) \cdot 12_4$;
- $2011_8 + 2012_8 - 4321_8$;
- $20671_8/131_8 - 137_8$;
- $120111_3/102_3 + 201_3 \cdot 12_3 - 11220_3$;
- $232011_5/104_5 + 1234_5 \cdot 322_5 - 1022131_5$;
- $3215_7 \cdot 24_7 - 11461_7/25_7 + 1532_7 - 115044_7$.

2.7. Записати в десятковій системі числення такі числа:

а) 100001101_2;
б) 7563401_8;
в) $(10)7(11)9_{12}$;
г) $0,221_3$;
д) $11001,11001_2$;
е) $437,3201_8$.

2.8. Перевести з однієї системи числення в іншу. Зробити перевірку.

а) $124_{10} \to x_4$;
б) $1340_{10} \to x_{14}$;
в) $101000_3 \to x_4$;
г) $32014_5 \to x_8$;
д) $33311_7 \to x_3$;
е) $34(11)57_{12} \to x_{11}$.

2.9. Перевести з однієї системи числення в іншу.

а) $10111001101_2 \to x_4$;
б) $1330_4 \to x_2$;
в) $101765_8 \to x_2$;
г) $1201222011,2111_3 \to x_9$;
д) $3387656,3455_9 \to x_3$;
е) $1011100010101,1_2 \to x_{16}$.

2.10. Перевести число $2012,2012_{10}$ з десяткової системи числення в двійкову, трійкову та четвіркову системи числення.

2.11. Знайти x, якщо:

а) $201_x = 41_8$;
б) $203_x = 53_{10}$;
в) $106_x = 153_7$;
г) $236_x = 1240_5$;
д) $324_x = 10022_3$;
е) $364_x = 3001_4$;
є) $401_x = 265_7$;
ж) $541_x = 2014_6$.

2.12. Знайти частку від ділення числа $(62xy427)_{10}$ на 99_{10}, а також невідомі цифри x і y, якщо ділення виконується без остачі.

2.13. Знайти таке найменше натуральне число m, яке в десятковій системі числення закінчується цифрою 6, причому, якщо цю цифру записати на початку числа, то воно збільшиться в 4 рази.

2.14. Число $(42x4y)_{10}$ ділиться на 72_{10}. Знайти цифри x і y.

2.15. Нехай $2 \cdot (xyz)_{10} = (xyz)_g$. Знайти $(xyz)_{10}$ і g.

2.16. Знайти число n, якщо $n = (abc)_7 = (cba)_9$.

2.17. Знайти 8-цифрове число $n = (abcdefgh)_{10}$, якщо числа $(abcd)_{10}$ та $(efgh)_{10}$ відрізняються на одиницю.

§ 3. Регулярні мови і регулярні вирази

В цьому параграфі опишемо важливий клас формальних мов – регулярні мови і їх задання з допомогою регулярних виразів, а також розглянемо алгебраїчні закони для регулярних виразів. Ці закони багато в чому подібні до алгебраїчних законів арифметики, проте між арифметичними і регулярними виразами є і суттєві відмінності. Регулярні вирази тісно пов'язані зі скінченними автоматами, які ми розглядатимемо в наступному розділі. Їх можна також розглядати як „мову програмування" для опису деяких важливих додатків, наприклад, програм текстового пошуку чи компонентів компілятора.

Нехай X – довільний алфавіт. Визначимо *регулярну мову (множину) в алфавіті X* рекурсивно наступним чином:

1) порожня множина $\emptyset$ є регулярною мовою;
2) для кожної букви $a \in X$ множина $\{a\}$ є регулярною мовою;
3) якщо A і B є регулярними мовами, то множини $A \cup B$, AB і A^* також є регулярними мовами;
4) інших регулярних мов не існує.

3.1. Приклад. Безпосередньо з означення випливає:

1) множина $\{e\}$ є регулярною мовою, бо $\{e\} = \emptyset^*$;
2) множина $\{001, 110\}$ – регулярна мова над бінарним алфавітом, бо $\{001, 110\} = (\{0\}\{0\}\{1\}) \cup (\{1\}\{1\}\{0\})$;
3) аналогічно, як і в пункті 2), можна показати, що будь-яка скінченна мова є регулярною.

Щоб спростити запис регулярних мов, визначимо поняття *регулярного виразу над алфавітом X* наступним чином:

1) порожня множина $\emptyset$ є регулярним виразом, який позначає порожню множину;
2) e є регулярним виразом, який позначає мову $\{e\}$;
3) a – регулярний вираз, що позначає мову $\{a\}$ для кожної букви $a \in X$;
4) якщо r_A і r_B є регулярними виразами, що позначають мови A і B відповідно, то $(r_A) + (r_B)$, $(r_A)(r_B)$ і $(r_A)^*$ є регулярними виразами, що позначають мови $A \cup B$, AB і A^* відповідно;
5) інших регулярних виразів над алфавітом X не існує.

Якщо r – регулярний вираз, то через $L(r)$ позначатимемо відповідну йому регулярну мову. Щоб зменшити кількість дужок в регулярних виразах, будемо вважати, що найвищий пріоритет має операція ітерації $*$, потім виконується конкатенація і останньою – операція $+$. Наприклад, запис регулярного виразу $((0)(1))|((1)(0))$, який позначає мову $\{01, 10\}$, спрощується до $01|10$, вираз 0^* позначає $\{0\}^*$, $(0|1)^*$ – $\{0, 1\}^*$, а $(0|1)^*001$ позначає множину всіх слів, які складаються з нулів і одиниць та закінчуються словом 001.

Зрозуміло, що для кожної регулярної мови можна знайти принаймні один регулярний вираз, який її позначає. І навпаки, для кожного регулярного виразу можна побудувати регулярну мову, що задається цим виразом. На жаль, для регулярної мови існує багато виразів. Наприклад, вирази $0^*1 + \emptyset$ і 0^*1 позначають одну й ту ж мову $\{0\}^*\{1\}$. Кажемо, що два регулярних вирази рівні $(=)$, якщо вони позначають одну і ту ж регулярну мову.

3.2. Твердження. *Нехай α, β і γ – регулярні вирази. Тоді:*

- $\alpha + \beta = \beta + \alpha$;
- $\emptyset^* = e$;
- $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$;
- $\alpha(\beta\gamma) = (\alpha\beta)\gamma$;
- $\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$;
- $(\alpha + \beta)\gamma = \alpha\gamma + \beta\gamma$;
- $\alpha e = e\alpha = \alpha$;
- $\emptyset\alpha = \alpha\emptyset = \emptyset$;
- $\alpha^* = \alpha + \alpha^*$;
- $(\alpha^*)^* = \alpha^*$;
- $\alpha + \alpha = \alpha$;
- $\alpha + \emptyset = \alpha$.

Доведення. Нехай α і β позначають мови A і B відповідно. Тоді $\alpha + \beta$ позначає $A \cup B$, а $\beta + \alpha$ позначає $B \cup A$. Але $A \cup B = B \cup A$ за означенням об'єднання, тому $\alpha + \beta = \beta + \alpha$. Решту рівностей доводяться аналогічно. □

Для зручності запису введемо позначення: $r^+ = rr^* = r^*r$.

Розглянемо деякі приклади.

3.3. Приклад. Довести рівності:

а) $a^*(a+b)^* = (a+ba^*)^*$;
б) $(ba)^+(a^*b^*+a^*) = (ba)^*ba^+b^*$.

а) Покажемо, що обидві частини рівності дорівнюють $(a+b)^*$. Дійсно, обидві частини рівності є підмножинами $(a+b)^*$, бо $(a+b)^*$ позначає множину всіх слів в алфавіті $\{a,b\}$. Таким чином, достатньо показати, що обидві частини містять $(a+b)^*$. Оскільки $e \in a^*$, то $a^*(a+b)^* \supset (a+b)^*$. З того, що $b \in ba^*$ випливає потрібне включення $(a+b)^* \subset (a+ba^*)^*$.

б) $(ba)^+(a^*b^*+a^*)=(ba)^*(ba)a^*(b^*+e)=(ba)^*ba^+b^*$.

3.4. Приклад. Регулярний вираз

$$1((0+1+2+3+4)(0+1+2+3+4+5+6+7+8+9)+5(0+1+2))$$

позначає множину натуральних чисел відрізка $[100, 152]$.

3.5. Приклад. Знайти регулярний вираз для множини всіх слів в алфавіті $\{0,1,2\}$, які є записами в трійковій системі чисел невід'ємних цілих степенів числа 9.

В трійковій системі числення степені 9^n запишуться як $1\underbrace{00\ldots0}_{2n}$. Таким чином, шуканий регулярний вираз має вигляд $1(00)^*$.

3.6. Приклад. Записати регулярний вираз для множини всіх слів у двійковому алфавіті, які містять підслово 001.

Кожне таке слово запишеться у вигляді $\alpha 001\beta$, де α і β – довільні слова в двійковому алфавіті. Таким чином, одержуємо шуканий регулярний вираз:

$$(0+1)^*001(0+1)^*.$$

3.7. Приклад. Записати регулярний вираз для множини всіх слів у двійковому алфавіті, які не містять підслова 001.

Якщо слово ω належить шуканій множині, то воно не містить підслова 00, за винятком того випадку, коли воно має суфікс 0^k для $k \geq 2$. Аналогічно, як і у прикладі 1.3, можна показати, що множина слів, які не містять підслова 00, записується з допомогою регулярного виразу $(01+1)^*(e+0)$. Таким чином, множина всіх

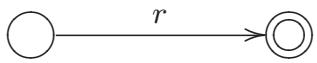
слів, які не містять підслова 001, запишеться наступним регулярним виразом:

$$(01+1)^*(e+0+000^*) = (01+1)^*0^*.$$

Розглянемо рівняння $\chi = \alpha\chi + \beta$, де α і β регулярні вирази в алфавіті X і $\chi \notin X$. З твердження 1.5 випливає, що $\chi = \alpha^*\beta$ – єдиний розв'язок рівняння, якщо $e \notin \alpha$. У випадку $e \in \alpha$ рівняння має безліч розв'язків, кожен з яких, очевидно, має вигляд $\alpha^*(\beta \cup \gamma)$, де γ – довільна (не обов'язково регулярна) мова. Використовуючи твердження 1.5, можна розв'язувати системи рівнянь з регулярними коефіцієнтами (методом, аналогічним методу Гауса для розв'язування систем лінійних рівнянь).

Регулярні вирази часто зображують *поміченими орієнтованими графами*, стрілки яких позначають буквами з алфавіту $X \cup \{e\}$. Для довільного регулярного виразу r його граф утворюється наступним чином:

Спочатку малюємо дві спеціальні вершини, які називаються початковою і кінцевою вершинами, і сполучаємо їх стрілкою, позначеною r:



Далі повторюємо наступні кроки доти, доки кожна позначка довільної стрілки не міститиме символів $+$, $\cdot$ і *:

1) замінюємо кожну стрілку з позначкою $f+g$ на дві паралельні стрілки з позначками f і g:

2) замінюємо кожну стрілку з позначкою fg на додаткову вершину і дві стрілки з позначками f і g:

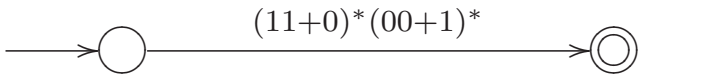
3) замінюємо кожну стрілку з позначкою f^* на додаткову вершину і на три стрілки з позначками e, f і e:

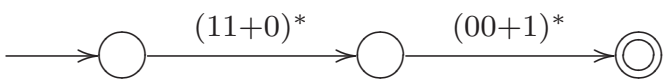
Через $G(r)$ позначимо помічений орієнтований граф регулярного виразу r. Очевидно, що кожна стрілка в $G(r)$ має позначку з множини $X \cup \{e\}$. Кожному шляху в $G(r)$ відповідає слово, яке одержується конкатенацією всіх букв, які позначають стрілки шляху. Слово x належить мові $L(r)$ тоді і тільки тоді, коли існує шлях в $G(r)$ з початкової вершини в кінцеву вершину, якому відповідає слово x. Кожна e-стрілка в $G(r)$, яка є єдиною вихідною стрілкою з некінцевої вершини чи єдиною вхідною стрілкою в непочаткову вершину, може бути стягнута в одну вершину.

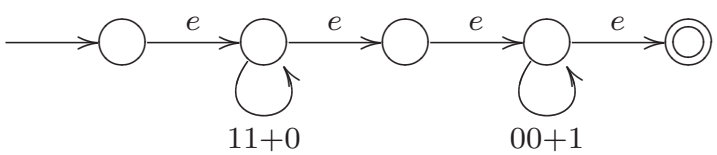
3.8. Приклад. Побудуємо граф $G(r)$ регулярного виразу

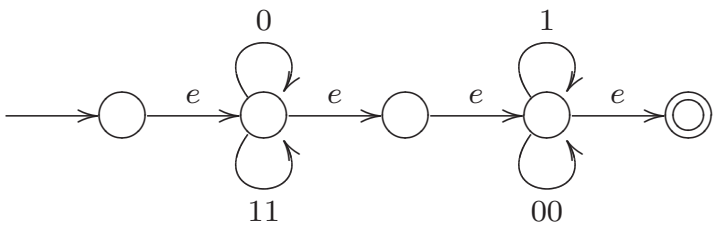
$$r = (11 + 0)^*(00 + 1)^*.$$

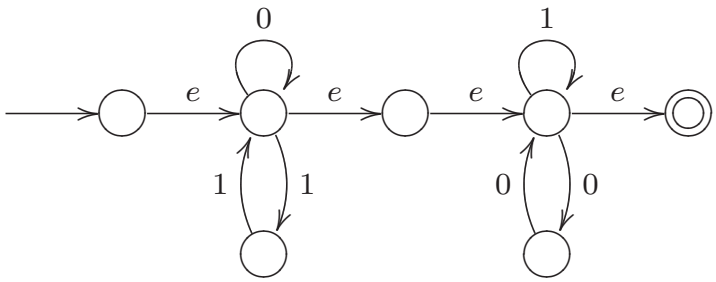
Зобразимо крок за кроком побудову даного графу:



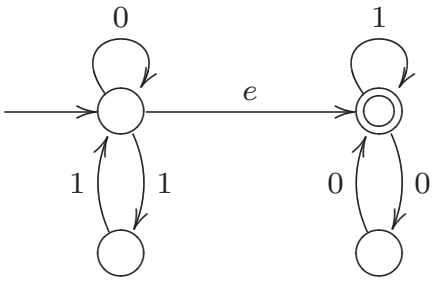








Стягнувши, де це можливо, e-стрілки, остаточно одержуємо помічений граф $G(r)$ регулярного виразу $r = (11 + 0)^*(00 + 1)^*$:



Рекомендована література : [**2**, с. 124–131], [**4**, с. 490–494], [**5**, с. 351–353], [**6**], [**15**, с. 13–31].

Питання та вправи до параграфа 3.

3.1. Яка формальна мова називається регулярною?

3.2. Де застосовуються регулярні вирази?

3.3. Опишіть процес побудови графа регулярного виразу.

3.4. Знайти слово найменшої довжини в кожній з наступних регулярних мов, заданій регулярним виразом. Які непорожні слова є найкоротшими в даних мовах?

а) $10 + (0 + 11)0^*1$;
б) $(00 + 11 + (01 + 10)(00 + 11)^*(01 + 10))^*$;
в) $((00 + 11)^* + (001 + 110)^*)^*$.

3.5. Визначити, чи належить рядок 1011 регулярним мовам, заданим регулярними виразами:

а) 10^*1;
б) $1(01)^*1^*$;
в) $(10)^+11$;
г) $(10)^+1011$;
д) $1(00)^*(11)^*$;
е) $(1 + 00)(01 + 0)1^*$.

3.6. Спростити регулярні вирази:

а) $(00)^*0 + (00)^*$;
б) $(0+1)(e+00)^+ + (0+1)$;
в) $(0+e)0^*1$.

3.7. Доведіть рівність $(0^2+0^3)^* = (0^20^*)^*$.

3.8. Чи є мова $\{0^{3m+4n} \mid m,n \geq 0\}$ регулярною? Відповідь обґрунтуйте.

3.9. Визначити алфавіт та побудувати регулярний вираз для наступних мов:

а) множина всіх непарних натуральних чисел, записаних в четвірковій системі числення;
б) множина всіх непарних натуральних чисел, записаних в трійковій системі числення.

3.10. Побудуйте регулярний вираз для множини всіх цілих чисел відрізка [10, 2012].

3.11. Розв'язати системи рівнянь з регулярними коефіцієнтами.

а) $\begin{cases} X_1 = e + 0X_1 + 1X_2 \\ X_2 = e + 1X_2 \end{cases}$

б) $\begin{cases} X_1 = 0X_2 + 1X_1 + e \\ X_2 = 0X_3 + 1X_2 \\ X_3 = 0X_1 + 1X_3 \end{cases}$

в) $\begin{cases} X_1 = (01^* + 1)X_1 + X_2 \\ X_2 = 11 + 1X_1 + 00X_3 \\ X_3 = e + X_1 + X_2 \end{cases}$

г) $\begin{cases} X_1 = 0X_3 + 1X_2 \\ X_2 = e + 1X_1 + 0X_3 \\ X_3 = e + 0X_1 \end{cases}$

3.12. Побудувати помічений граф для наступних регулярних виразів:

а) $01 + 11^*$;
б) $(00+10)(101)^* + 01$;
в) $((00+11)^* + (001+110)^*)^*$;
г) $a^*b(c + da^*b)^*$;
д) $(a + bc^*d)^*bc^*$.

3.13. Побудувати регулярні вирази для мови всіх слів в алфавіті $\{0, 1\}$, для яких виконується наступна умова:

а) містять підслово 000 або 111;
б) в яких п'ятою буквою справа є 0;
в) містять не більше однієї пари послідовних одиниць;
г) кількість нулів кратна п'яти;
ґ) містять непарну кількість букв 0;
д) не містять підслова 000;
е) не містять ні підслова 000, ні 111;
є) не містять підслова 011;
ж) містять непарну кількість входжень підслова 011;
з) кількість одиниць ділиться на п'ять, а кількість нулів парна.

3.14. Нехай L – мова над алфавітом $\{0\}$. Довести, що мова L^* є регулярною. [Підказка: доведіть і використайте той факт, що якщо a та b є взаємно простими натуральними числами, то для кожного натурального $n \geq ab$ існують такі невід'ємні цілі числа u та v, що виконується рівність $n = ua + vb$.]

§ 4. Формальні породжуючі граматики

Ми визначили формальну мову L як множину слів скінченної довжини в алфавіті X. Якщо L містить скінченну кількість слів, то найбільш очевидний спосіб описати мову – скласти перелік всіх її слів. Однак для багатьох мов не можна (або, можливо, не бажано) обмежувати довжини слів. Отже, доводиться розглядати мови, які містять як завгодно багато слів. Відразу виникає важливе питання: як описати мову L в тому випадку, коли вона нескінченна. Очевидно, що такі формальні мови неможливо визначити вичерпним переліком слів, які їм належать, і тому потрібно шукати інші способи їх опису. Як і раніше, ми хочемо, щоб опис мови був скінченним, хоча сама мова може бути й нескінченною. Відомо декілька методів опису мов, які задовольняють цю вимогу. Один з них показаний в попередньому параграфі для регулярних мов. Розглянемо метод, який полягає в використанні структури, яка називається *граматикою*.

Формальною породжуючою граматикою називається четвірка виду $G = (N, T, S, P)$, де N і T– скінченні непорожні неперетинні множини, елементи яких будемо називати *нетермінальними та термінальними символами* відповідно, $S \in N$ – *початковий нетермінальний символ*, P – скінченна множина *продукцій (правил перетворення)* вигляду $\xi \to \eta$, де ξ та η – слова в алфавіті $N \cup T$. Букви термінального алфавіту позначатимемо малими латинськими буквами чи цифрами, а нетермінального алфавіту – великими латинськими буквами.

Нас цікавитимуть слова, які можуть бути породжені продукціями граматики. Нехай $G = (N, T, S, P)$ – граматика і $\alpha_0 = \sigma\xi\tau$ (тобто α_0 – конкатенація слів σ, ξ і τ), $\alpha_1 = \sigma\eta\tau$ – слова в алфавіті $N \cup T$. Якщо $\xi \to \eta$ – продукція граматики G, то кажуть, що α_1 *безпосередньо виводиться* з α_0 і записують $\alpha_0 \Rightarrow \alpha_1$. Якщо ж $\alpha_0, \alpha_1, \ldots, \alpha_n$ – слова в алфавіті $N \cup T$ такі, що $\alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_{n-1} \Rightarrow \alpha_n$, то говорять, що α_n *виводиться* з α_0 і записують $\alpha_0 \Rightarrow^* \alpha_n$.

Мовою, породженою граматикою $G = (N, T, S, P)$, називають множину всіх слів, які виводяться з початкового символу S і містять тільки термінальні символи. Її позначають $L(G)$. Таким чином,

$$L(G) = \{\omega \in T^* \mid S \Rightarrow^* \omega\}.$$

4.1. Приклад. Граматика G, яка породжує речення (слово з чотирьох букв формальної мови $L(G)$) „`хороший студент одержав п'ятірку`” матиме вигляд $G = (N, T, S, P)$, де

N={<речення>, <група підмета>, <група присудка>, <означення>, <підмет>, <присудок>, <додаток>};

T={ хороший, студент, одержав, п'ятірку };

S=<речення>.

Множина продукцій P містить продукції:

<речення> → <група підмета> <група присудка>,

<група підмета> → <означення> <підмет>,

<група присудка> → <присудок> <додаток>,

<означення> → хороший,

<підмет> → студент,

<присудок> → одержав,

<додаток> → п'ятірку.

Розглянемо виведення даної граматики з початкового нетермінального символу:

<речення> ⇒ <група підмета> <група присудка> ⇒
<означення> <підмет> <група присудка> ⇒
<означення> <підмет> <присудок> <додаток> ⇒
хороший <підмет> <присудок> <додаток> ⇒
хороший студент <присудок> <додаток> ⇒
хороший студент одержав <додаток> ⇒
хороший студент одержав п'ятірку.

Таким чином, $L(G)$ складається з єдиного слова (речення української мови), яке містить чотири термінали.

$$L(G) = \{\text{„хороший студент одержав п'ятірку”}\}.$$

4.2. Приклад. Нехай G – граматика з нетермінальним алфавітом $N = \{S, A\}$, множиною терміналів $T = \{a, b\}$, початковим символом S і множиною продукцій $P = \{S \to bA, S \to a, A \to bb\}$. Знайдемо мову $L(G)$, що породжується цією граматикою.

Із початкового символу S можна вивести слово bA за допомогою продукції $S \to bA$ чи застосувати продукцію $S \to a$, щоб вивести a. З bA, скориставшись продукцією $A \to bb$, можна вивести слово bbb. Оскільки інших виведень не існує, то $L(G) = \{a, bbb\}$.

4.3. Приклад. Розглянемо граматику $G = (N = \{S\}, T = \{0, 1\}, S, P = \{S \to e, S \to 0S1\})$ і знайдемо мову, що породжується нею.

Граматика G містить єдиний нетермінальний символ і єдине правило $S \to 0S1$, права частина якого містить цей символ. Таким чином, єдиними виведеннями граматики G є виведення форми:

$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow \ldots \Rightarrow 0^nS1^n \Rightarrow 0^ne1^n = 0^n1^n$$

Отже,

$$L(G) = \{0^n1^n \mid n \geq 0\}.$$

Якщо є багато правил граматики з однаковими лівими частинами

$$A \to x_1, A \to x_2, \ldots A \to x_m,$$

то їх об'єднують в одне правило наступного вигляду:

$$A \to x_1 \mid x_2 \mid \ldots \mid x_m.$$

Наприклад, правила з попереднього прикладу можуть бути записані як $S \to e \mid 0S1$.

4.4. Приклад. Розглянемо граматику $G = (N = \{S, A, B\}, T = \{a, b\}, S, P)$, де множина продукцій P містить наступні правила:

$$S \to ABA,\ A \to a \mid bb,\ B \to bS \mid e,$$

і знайдемо мову $L(G)$.

Оскільки нетермінальний символ A може тільки продукувати термінальні символи, то його заміщення слід відкласти наостанок. Таким чином, виведення граматики G мають наступну загальну форму:

$S \Rightarrow ABA \Rightarrow AbSA \Rightarrow AbABAA \Rightarrow AbAbSAA \Rightarrow A(bA)^2BA^3 \Rightarrow \ldots \Rightarrow A(bA)^nBA^{n+1} \Rightarrow A(bA)^nA^{n+1}$.

Насамкінець замінюємо нетермінал A терміналами a та bb, і одержуємо слова вигляду:

$$(a + bb)(ba + bbb)^n(a + bb)^{n+1}, n \geq 0.$$

Таким чином, $L(G)$ складається з усіх побудованих вище слів.

Граматики класифікують за типами продукцій. Розглянемо класифікацію, яку запропонував американський математик і лінгвіст Н. Хомський. Принцип цієї класифікації полягає в тому, що на продукції накладено певні обмеження. Він поділив граматики на чотири типи.

У граматиці *типу 0* немає жодних обмежень на продукції. Граматика *типу 1* може мати лише продукції вигляду $\xi \to \eta$, де довжина слова η не менша, ніж довжина слова ξ, або у вигляді $\xi \to e$. У граматиці *типу 2* можуть бути лише продукції $A \to \omega$, де A – нетермінальний символ, $\omega \in (N \cup T)^*$. Граматика *типу 3* може мати такі продукції: $A \to \omega B$, $A \to \omega$ і $A \to e$, де A, B – нетермінали, ω – слово з терміналів. Із цих означень випливає, що кожна граматика типу n, де $n = 1, 2, 3$, є граматикою типу $n - 1$. Граматики типу 2 називають *контекстно вільними*, бо нетермінал A в лівій частині продукції $A \to \eta$ можна замінити словом η в довільному оточенні щоразу, коли він зустрічається, тобто незалежно від контексту. Мову, яку породжує граматика типу 2, називають *контекстно вільною*. Якщо в множині продукцій P є продукція виду $\gamma\mu\delta \to \gamma\nu\delta$, $|\mu| \leq |\nu|$, то граматику називають *контекстно залежною*, оскільки μ можна замінити на ν лише в оточенні слів $\gamma \ldots \delta$, тобто у відповідному контексті. Мову, яку породжує граматика типу 1, називають контекстно залежною. Граматику типу 3 називають *праволінійною*.

Двоїсто визначають *ліволінійну граматику*; вона може мати такі продукції: $A \to B\omega$, $A \to \omega$ і $A \to e$, де A, B – нетермінали, ω – слово з терміналів. Ліволінійні і праволінійні граматики називаються *регулярними*. В параграфі 5 наступного розділу ми доведемо, що регулярні граматики породжують регулярні мови і тільки їх.

4.5. Приклад. Побудуємо контекстно вільну граматику, яка породжує мову $L = \{0^n 1^{2n} \mid n \geq 0\}$.

Ця мова нагадує мову, породжену граматикою з прикладу 4.2, в тому сенсі, що замінивши кожну 1 на 11, одержимо граматику для L. Отже, шукана граматика має вигляд

$$G = (\{S\}, \{0, 1\}, S, \{S \to e \mid 0S11\}).$$

Іншим методом побудови граматики є відшукання рекурсивної функції, яка пов'язує довші слова мови L з коротшими словами. Наприклад, слово $0^{n+1}1^{2(n+1)}$ можна записати як $0(0^n 1^{2n})11$. Таким чином, якщо існує виведення $S \Rightarrow^* 0^n 1^{2n}$, то потрібна продукція $S \to 0S11$, щоб одержати виведення $S \Rightarrow^* 0^{n+1}1^{2(n+1)}$. Продемонструємо цю ідею в наступному прикладі.

4.6. Приклад. Побудуємо контекстно вільну граматику, що породжує мову $L = \{x \in \{0, 1\}^* \mid x = x^R\}$.

Оскільки k-та зліва буква кожного слова має дорівнювати k-тій справа букві, то слово $x \in L$ довжини $2n + 2$ може бути записане або як $0y0$, або $1y1$ для деякого слова $y \in L$ довжини $2n$. Отже, нам потрібні правила $S \to 0S0$ і $S \to 1S1$, щоб породити слово x рекурсивно. Зі сказаного вище випливає, що мова L породжується граматикою $G = (\{S\}, \{0, 1\}, S, P)$, де множина P містить наступні продукції:

$$S \to e \mid 0 \mid 1 \mid 0S0 \mid 1S1.$$

З чотирьох класів граматик ієрархії Хомського клас контекстно вільних граматик є найбільш важливим з точки зору застосувань до мов програмування і компіляції. З допомогою контекстно вільної граматики можна визначити більшу частину синтаксичної структури мов програмування. Крім того, вона є основою різних схем задання перекладів на машинну мову комп'ютера.

Слово ω належить мові $L(G)$ контекстно вільної граматики G тоді і лише тоді, коли існує виведення $S \Rightarrow^* \omega$ в G. Воно показує,

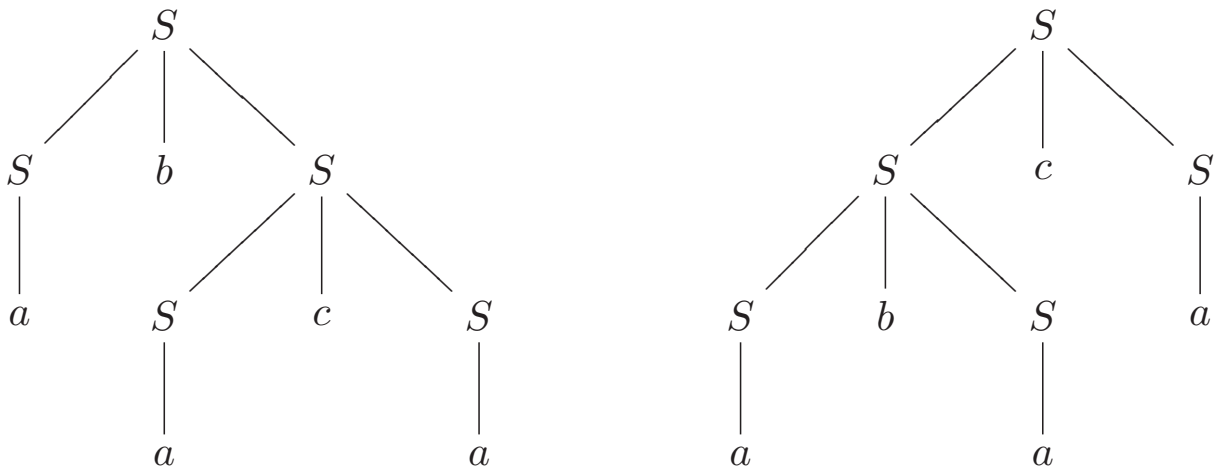
як крок за кроком застосовуються продукції граматики, щоб одержати слово ω з початкового нетермінального символу S. Це виведення можна також продемонструвати наочно з допомогою дерева, яке називатимемо *деревом виведення*. Наприклад, розглянемо контекстно вільну граматику $G = (\{S\}, \{a, b, c\}, S, P)$, де $P = \{S \to SbS \mid ScS \mid a\}$ і слово $abaca \in L(G)$. Виведення

$$S \Rightarrow SbS \Rightarrow SbScS \Rightarrow abScS \Rightarrow abSca \Rightarrow abaca$$

показано на рисунку (а), а виведення

$$S \Rightarrow ScS \Rightarrow SbScS \Rightarrow abScS \Rightarrow abSca \Rightarrow abaca$$

– на рисунку (б).

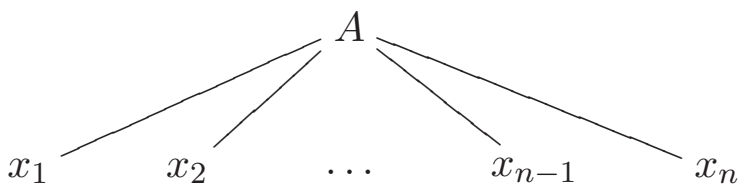


(а) (б)

В загальному випадку для контекстно вільної граматики $G = (N, T, S, P)$ дерево виведення будується наступним чином:

1. Вершиною дерева є початковий нетермінальний символ S.

2. Повторюємо наступні кроки доти, доки кожен листок дерева буде або порожнім символом e, або термінальним символом: для кожного листка $A \in N$ дерева обираємо продукцію $A \to x_1 x_2 \dots x_n$ граматики G, де $x_i \in N \cup T$, і замінюємо вершину $A \in N$ деревом:



Якщо конкатенація листків дерева виведення є словом $\omega \in T^*$, то кажемо, що дане дерево є *деревом виведення для слова* ω.

Дерево виведення слова ω відповідає виведенню ω в граматиці G. Часто дерево виведення відповідає більш ніж одному виведенню слова ω. Наприклад, для граматики G, визначеної вище, є 16 різних виведень слова $\omega = abaca$ в граматиці G. Серед них 8 виведень, які починаються з $S \Rightarrow SbS$, зображуються одним і ти же деревом виведення, показаним на рисунку (а). Решту 8 виведень, що починаються з $S \Rightarrow ScS$, зображені деревом виведення на рисунку (б).

Щоб зробити відповідність між деревами виведення і виведеннями в граматиці G взаємно однозначною, назвемо виведення слова $\omega \in L(G)$ *лівим виведенням*, якщо завжди застосовується відповідна продукція граматики G для заміни найлівішого нетермінального символу в проміжному слові при виведенні слова ω. Очевидно, що кожне дерево виведення слова ω відповідає єдиному лівому виведенню слова ω. Наприклад, серед восьми виведень для $\omega = abaca$, які починаються з $S \Rightarrow SbS$, єдиним лівим виведенням є

$$S \Rightarrow SbS \Rightarrow abS \Rightarrow abScS \Rightarrow abacS \Rightarrow abaca.$$

Аналогічно дається означення правого виведення.

Контекстно вільна граматика $G = (N, T, S, P)$ називається *неоднозначною*, якщо існує слово $\omega \in L(G)$, яке має два або більше різних дерев виведень. Якщо граматика використовується для визначення мови програмування, то бажано, щоб вона була однозначною. В іншому випадку програміст і компілятор можуть порізному зрозуміти смисл деяких програм.

4.7. Приклад. Найвідомішим прикладом неоднозначності в мовах програмування є „плаваюче” `else`.

Розглянемо граматику G з правилами

$$S \to \texttt{if}\ b\ \texttt{then}\ S\ \texttt{else}\ S \mid \texttt{if}\ b\ \texttt{then}\ S \mid a.$$

Ця граматика неоднозначна, оскільки слово

$$\texttt{if}\ b\ \texttt{then if}\ b\ \texttt{then}\ a\ \texttt{else}\ a$$

має два виводи

$$\texttt{if}\ b\ \texttt{then}\ (\texttt{if}\ b\ \texttt{then}\ a)\ \texttt{else}\ a \text{ та}$$

$$\texttt{if } b \texttt{ then } (\texttt{if } b \texttt{ then } a \texttt{ else } a),$$

яким відповідають два різних дерева виведення.

Визначена нами неоднозначність – це властивість граматики, а не мови. Для деяких неоднозначних граматик можна побудувати рівносильні їм однозначні граматики.

4.8. Приклад. Розглянемо граматику і мову з попереднього прикладу. Граматика G неоднозначна, бо `else` можна асоціювати з двома різними `then`. З тієї ж причини мови програмування, в яких дозволяються як оператори виду `if-then`, так і оператори виду `if-then-else`, можуть бути неоднозначними. Невизначеність можна виправити, якщо домовитися, що `else` повинно асоціюватися з останнім з записаних перед ним `then`.

Можна поправити граматику з попереднього прикладу, ввівши два нетермінали S_1 і S_2 і вимагаючи, щоб S_2 породжував оператори виду `if-then-else`, тоді як S_1 може породжувати оператори обох видів. Продукції нової граматики є такими:

$$S_1 \to \texttt{if } b \texttt{ then } S_1 \mid \texttt{if } b \texttt{ then } S_2 \texttt{ else } S_1 \mid a$$

$$S_2 \to \texttt{if } b \texttt{ then } S_2 \texttt{ else } S_2 \mid a.$$

Той факт, що перед `else` знаходиться тільки S_2, гарантує появу всередині конструкції `then-else` або букви a, або іншого `else`. Таким чином, структура $\texttt{if } b \texttt{ then } (\texttt{if } b \texttt{ then } a) \texttt{ else } a$ не виникає.

Рекомендована література : [**1**, с. 92–150], [**2**, с. 105–123, 163–192], [**5**, с. 341–351, 354–359], [**16**, с. 278–285], [**23**, с. 14–24].

Питання та вправи до параграфа 4.

4.1. Дайте означення граматики.

4.2. Які типи граматик ви знаєте?

4.3. Як будується дерево виведення для слова ω?

4.4. Чи може для деякого слова ω існувати декілька дерев виведення?

4.5. Нехай $N = \{S, A, B\}$, $T = \{a, b\}$. Визначити мову, породжену граматикою $G = (N, T, S, P)$ з вказаною множиною продукцій P:

а) $P = \{S \to AB \mid aA, A \to a, B \to ba\}$;
б) $P = \{S \to AB \mid AA, A \to aB \mid ab, B \to b\}$;
в) $P = \{S \to AA \mid B, A \to aaA \mid aa, B \to bB \mid b\}$;
г) $P = \{S \to aS \mid Sb \mid a\}$;
д) $P = \{S \to SS \mid a \mid b\}$;
е) $P = \{S \to SS \mid aSb \mid e\}$;
є) $P = \{S \to e \mid aS \mid aSbS\}$;
ж) $P = \{e \mid aSbS \mid bSaS\}$.

4.6. Знайти мову, породжену граматикою $G = (N, T, S, P)$, де $N = \{S, A, Q, R\}$, $T = \{a\}$, а множина продукцій $P = \{S \to QAQ, QA \to QR, RA \to AAR, RQ \to AAQ, A \to a, Q \to e\}$.

4.7. Нехай задана граматика $G = (N, T, S, P)$, де $N = \{S, A\}$, $T = \{0, 1\}$, множина продукцій

$$P = \{S \to 0S \mid 1A \mid 1 \mid e, A \to 1A \mid 1\}.$$

а) Побудувати вивід для слова $0^3 1^6$;
б) Показати, що дана граматика породжує мову $\{0^m 1^n \mid m, n = 0, 1, 2, \ldots\}$.

4.8. Побудувати граматики, які породжують вказані мови:

а) $\{01^{2n} \mid n = 0, 1, \ldots\}$;
б) $\{0^{3n} 1^n \mid n = 0, 1, \ldots\}$;
в) $\{0^n 1^m 0^n \mid m, n = 0, 1, \ldots\}$.

4.9. З'ясувати, до яких типів за класифікацією Хомського належить граматика G з множиною продукцій P:

а) $P = \{S \to SAB, SA \to a, B \to b\}$;
б) $P = \{S \to ABS, AB \to ab, S \to c\}$;
в) $P = \{S \to aAB, A \to Bb, B \to e\}$;
г) $P = \{S \to aA, A \to bB, B \to b \mid e\}$;
д) $P = \{S \to A, A \to B, B \to e\}$.

4.10. Побудувати ліволінійну граматику для формальної мови всіх слів довжини менше шести в алфавіті $\{0, 1, m, n\}$, які починаються з букв m або n.

4.11. Побудувати праволінійну граматику, яка породжує множину цілих чисел.

4.12. Довести, що жодне слово мови $L(G)$ не містить підслова ba, де контекстно-вільна граматика G задана продукціями

$$S \to aS \mid bA \mid a, \;\; A \to bA \mid b.$$

4.13. Показати, що кожне слово мови $L(G)$ містить більше букв a, ніж букв b, де контекстно-вільна граматика G задана продукціями

$$S \to Sa \mid bSS \mid SSb \mid SbS \mid a.$$

4.14. Показати, що мова

$$\{\omega \in \{0,1\}^* \mid \omega \text{ містить однакову кількість букв 0 та 1 }\}$$

не породжується граматикою з продукціями

$$S \to 0S1 \mid 01S \mid 1S0 \mid 10S \mid S01 \mid S10 \mid e.$$

4.15. Довести, що мова

$$\{\omega \in \{0,1\}^* \mid \omega \text{ містить однакову кількість букв 0 та 1 }\}$$

породжується граматикою з продукціями

$$S \to SS \mid 0S1 \mid 1S0 \mid e.$$

4.16. Яку формальну мову породжує граматика з продукціями

$$S \to aSBa|aba, \quad aB \to Ba, \quad bB \to bb \;\; ?$$

4.17. Нехай G – граматика з $N = \{S\}$, $T = \{a, b, c\}$, початковий символ S і множина продукцій

$$P = \{S \to abS \mid bcS \mid bbS \mid a \mid cb\}.$$

Побудувати дерево виведення для слів:

а) bbbcbba
б) bcabbbbbcb.

4.18. Показати, що граматика

$$G = \{\{S\}, \{+, x\}, S, \{S \to S + S \mid x\}\}$$

є неоднозначною. Побудувати всі дерева виведення для слів $x+x+x$ та $x + x + x + x$.

Розділ II

Скінченні автомати

§ 1. Автомати. Їх типи та задання

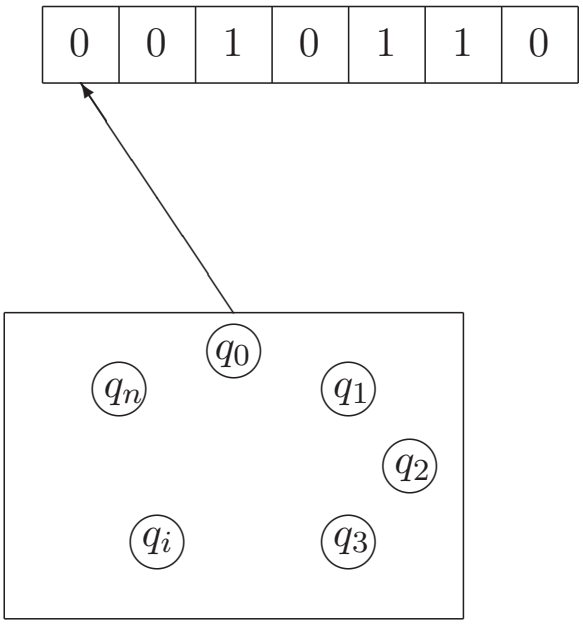
Існує багато різноманітних методів перетворення інформації. Алгоритм роботи перетворювачів інформації можна інтуїтивно визначити як деяку сукупність правил, за якими даний пристрій перетворює вхідну інформацію у вихідну. В даному параграфі ми розглянемо перетворювач інформації, який називається автоматом.

Щоб перейти до формальних методів опису процесу перетворення інформації з допомогою автоматів, розглянемо деякий пристрій, який має n входів і m виходів. Будемо вважати, що на входи інформація подається в дискретні моменти часу $t_0, t_1, \ldots, t_n$. Інтервал часу $t_{i+1} - t_i$ називається *тактом*. Аналогічно і з виходів інформація одержується дискретно в часі. На кожному із входів (виходів) сигнал може приймати дискретні значення з деякого набору $u_0, u_1, \ldots, u_q$. Сукупність значень вхідних сигналів у момент часу t_i називатимемо *вхідною буквою*, а послідовність вхідних букв – *вхідним словом*. Множину всіх вхідних букв назвемо *вхідним алфавітом*. Аналогічні означення відносяться і до вихідних сигналів.

Давши це означення, можна уявляти будь-який перетворювач інформації як пристрій з одним входом, на який надходять слова, складені з букв вхідного алфавіту, і з одним виходом, з якого отримуються слова в вихідному алфавіті. Процес перетворення інформації в такому пристрої зводиться до встановлення деякої відповідності між словами вхідного та вихідного алфавітів.

Автоматом називають перетворювач алфавітної інформації, який має один вхід і один вихід. Для задання умов функціонування автомата фіксуються три множини, елементами яких є букви трьох алфавітів: вхідного $X = \{x_1, \ldots, x_m\}$, вихідного $Y = \{y_1, \ldots, y_l\}$ і множини внутрішніх станів автомата $Q = \{q_0, \ldots, q_n\}$.

Автомат складається з трьох частин: *стрічки, головки і керуючого пристрою*. Стрічка використовується для запису вхідних даних (вхідного слова). Вона поділена на скінченну кількість комірок.



Головка переглядає стрічку, читає букву з неї і передає одержану інформацію керуючому пристрою. Кожного разу головка переглядає тільки одну комірку стрічки, читає записану в ній букву і переходить до наступної комірки праворуч. Керуючий пристрій складається з елементів множини станів Q.

Автомат функціонує в дискретні моменти часу, які прийнято позначати цілими невід'ємними числами $t = 0, 1, \ldots, k$. В початковий момент часу $t = 0$ керуючий пристрій завжди знаходиться в початковому стані $q_0 = q(0)$, в якому автомат починає роботу над вхідним словом. Потім за активним станом і зчитаною головкою буквою він визначає, в який стан потрібно перейти, і яку букву треба написати на місці прочитаної вхідної букви. На згаданих вище множинах задаються дві функції. *Функція переходів* δ, яка визначає стан автомата $q(t+1) \in Q$ в момент часу $t+1$ в залежності від його стану $q(t) \in Q$ і значення вхідного сигналу $x(t) \in X$ в момент часу t: $q(t+1) = \delta(q(t), x(t))$. Тобто зміна станів визначається за допомогою функції двох змінних $\delta : Q \times X \to Q$. *Функція виходів* $f : Q \times X \to Y$ визначає значення вихідного сигналу $y(t) \in Y$ в залежності від стану автомата $q(t) \in Q$ і значення вхідного сигналу $x(t) \in X$ в момент часу t. Таким чином, якщо керуючий пристрій

перебуває в стані $q \in Q$ і головка зчитує з стрічки букву $a \in X$, то керуючий пристрій переходить в новий стан $p = \delta(q, a)$ і на місці букви a записується буква $b = f(q, a)$.

Коли на вхід автомата подавати слово (послідовність вхідних букв), на виході також з'являється слово (послідовність вихідних букв). У цьому випадку автомат A можна розглядати як алфавітний перетворювач інформації, який перетворює слова вільної напівгрупи $F(X)$ в слова вільної напівгрупи $F(Y)$. Визначена таким чином словесна функція $g_A : F(X) \to F(Y)$ називається *функцією, визначеною автоматом A*, або *автоматною A-функцією*.

Для повного опису автомата потрібно виділити множину $F \subset Q$ *кінцевих станів*, в яких він підтверджує, що вхідне слово належить шуканій мові. Таким чином, ми приходимо до такого визначення автомата.

Сімка $A = (Q, X, Y, \delta, f, q_0, F)$ називається *автоматом Мілі*, якщо вона складається із множини станів автомата Q, множини вхідних букв X, множини вихідних букв Y, функції переходів $\delta : Q \times X \to Q$, функції виходів $f : Q \times X \to Y$, початкового стану q_0 та множини кінцевих станів F. Множини X і Y називають відповідно *вхідним і вихідним алфавітами автомата*. Якщо $\delta(a, x) = a'$, то кажуть, що автомат Q під дією вхідного сигналу $x \in X$ переходить в стан a', або сигнал x переводить автомат Q із стану a в стан a'. Якщо $f(a, x) = y$, то кажуть, що автомат Q перетворює в стані a вхідний сигнал $x \in X$ у вихідний сигнал $y \in Y$.

Як саме працює автомат на вхідному слові? На початку роботи керуючий пристрій перебуває в початковому стані q_0, на стрічці записане вхідне слово і головка зчитує букву з першої зліва комірки. Потім автомат працює крок за кроком відповідно до функцій переходів δ та виходів f. Він зупиняє роботу тоді, коли головка прочитає букву, записану в останній справа комірці стрічки. Коли обробка автоматом вхідного слова завершилася, то кажуть, що *автомат розпізнає вхідне слово*, якщо він зупиняється в одному з кінцевих станів множини F. В протилежному випадку, кажуть, що *вхідне слово не розпізнається автоматом*. Якщо ж множина F кінцевих станів не задана, то вважається, що автомат розпізнає всі слова вхідного алфавіту. Формальна мова всіх слів вхідного алфавіту X, які розпізнаються автоматом A, позначається $L(A)$. В цьому випадку кажуть, що мова $L(A)$ *розпізнається автоматом*

A. Множина всіх вихідних слів автомата A, які є результатом роботи автомата на мові $L(A)$, називається *мовою, що породжується автоматом* A.

Автомат $A = (Q, X, Y, \delta, f, q_0, F)$ називається *скінченним*, якщо всі три множини – Q, X і Y – скінченні, і *нескінченним*, якщо хоч одна з них нескінченна.

Автомат називається *повним* або *повністю визначеним*, якщо функції переходів і виходів всюди визначені, і *частковим*, якщо хоч одна з них є частково визначеною функцією.

Іноді трапляються автомати, в яких компонент δ – не функція, а деяке відношення, тобто в таких автоматах не виконується умова однозначності переходу. Автомати такого типу називаються *недетермінованими*. Якщо ж відношення δ є функцією, то автомат називається *детермінованим*. Отже, для детермінованого автомата Q з початковим станом a однозначно знаходиться стан b, в який автомат перейде під дією слова $\omega \in F(X)$. А в недетермінованому автоматі таких станів може бути не один, а декілька. Ясно, що клас детермінованих автоматів є підкласом класу недетермінованих автоматів.

Крім класів детермінованих і недетермінованих автоматів існують і інші спеціальні класи. Серед них виділяються деякі вироджені класи автоматів, в яких одна з множин (Q, X або Y) одноелементна. У таких випадках розглядаються спрощені моделі автоматів. Наведемо деякі приклади.

Автомат без пам'яті – це трійка (X, Y, f), де $f : X \to Y$. Цей автомат виконує побуквену трансформацію букв вхідного алфавіту X у букви вихідного алфавіту Y, при якій на одну і ту ж вхідну букву $x \in X$ він реагує одною і тою ж вихідною буквою $y \in Y$. Поняття *автомат без пам'яті* – основне в теорії перемикальних схем, яка вивчає способи подання таких автоматів мережами із логічних елементів.

Автономний автомат – це четвірка (Q, Y, δ, f), де $\delta : Q \to Q$, $f : Q \to Y$. Для такого автомата поданням початкового стану визначається весь подальший процес його функціонування.

Автомат без виходів – це п'ятірка (Q, X, δ, q_0, F), де F – множина кінцевих станів автомата, а $q_0 \in Q$ – початковий стан автомата. Такі автомати будуть детальніше розглянуті в наступних параграфах.

Із перелічених класів автоматів тільки автомати без виходів, з точки зору теорії, викликають інтерес, бо в двох попередніх випадках поведінка автомата є наперед визначеною.

Автомати Мура є окремим випадком автоматів Мілі. Автомат $A = (Q, X, Y, \delta, f, q_0, F)$ називається *автоматом Мура*, якщо $f(q, x_1) = f(q, x_2)$ для кожних $q \in Q$ та $x_1, x_2 \in X$. В автоматі Мура вихідний сигнал у момент часу t однозначно визначається станом автомата в той же момент часу, тобто $y(t) = h(q(t))$, де $h : Q \to Y$. Функція h називається *функцією відміток автомата*, а її значення $h(a)$ на стані a – *відміткою цього стану*.

Хоча автомати Мура і є окремим випадком автоматів Мілі, в теорії автоматів вони заслуговують на особливу увагу, оскільки в ряді випадків їх специфічні властивості дають можливість будувати більш змістовну і глибоку теорію, ніж теорія автоматів Мілі.

Перейдемо до розгляду скінченних автоматів і способів їх задання.

У загальному випадку для задання скінченних автоматів користуються двома стандартними способами, які називаються *універсальними* – це таблиці переходів і виходів та графи переходів і виходів.

Таблиці переходів і виходів автомата – це дві матриці однакової розмірності, стовпчики яких відмічені різними буквами вхідного алфавіту, а рядки – різними символами станів автомата. Для функції переходів (перша матриця) на перетині i-го рядка, відміченого станом $a \in Q$, і j-го стовпчика, відміченого буквою $x \in X$, знаходиться значення $\delta(a, x) = b$. Аналогічно для функції виходів (друга матриця) на перетині i-го рядка і j-го стовпчика знаходиться значення $f(a, x) = y \in Y$. Вважають, що початковий стан автомата відмічає перший рядок як в таблиці переходів, так і в таблиці виходів.

1.1. Приклад. Автомат A=$(\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{0, 1\}, \delta, f, q_0, \{q_1\})$ задається такими таблицями переходів і виходів:

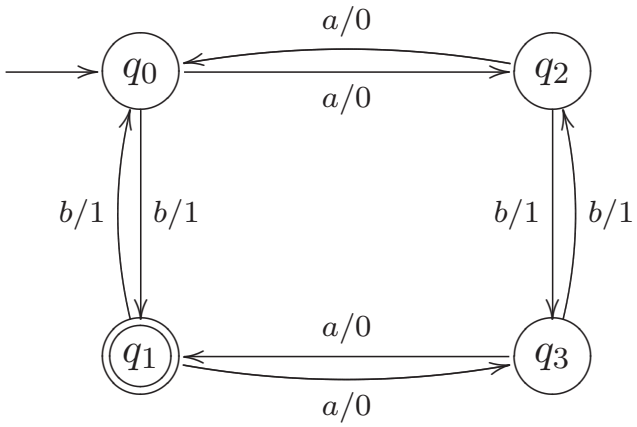
δ	a	b
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

f	a	b
q_0	0	1
q_1	0	1
q_2	0	1
q_3	0	1

В якому б стані даний автомат не перебував, він замінює букви a і b на 0 і 1 відповідно. Таким чином, скінченний автомат A переписує будь-яке слово в алфавіті $\{a, b\}$ в слово в алфавіті $\{0, 1\}$. Автомат A не є автоматом Мура, оскільки перебуваючи в стані q_0 він замінює букву a на 0, а букву b – на 1.

Граф переходів і виходів скінченного автомата A є альтернативним шляхом для зображення A. Він являє собою помічений орієнтований граф, вершини якого взаємно однозначно відповідають станам автомата. Стрілки графа відмічені парами букв: перша – буква вхідного алфавіту, друга – буква вихідного алфавіту. У даному випадку відповідність між графом переходів і виходів та функціями переходів δ і виходів f така, що коли $\delta(a, x) = a'$ і $f(a, x) = y$, то в графі з вершини a у вершину a' веде стрілка, відмічена парою (x/y), і навпаки, коли в графі переходів і виходів автомата існує стрілка (a, a'), відмічена парою (x/y), то для функцій переходу δ і виходу f виконуються рівності $\delta(a, x) = a'$ і $f(a, x) = y$. Крім того, щоб підкреслити початковий стан q_0, проводиться стрілка без початкової вершини, кінцевою вершиною якої є стан q_0. Кінцевий стан позначається двома концентричними колами.

1.2. Приклад. Граф переходів і виходів для автомата з прикладу 1.1 має вигляд:



З наведеної відповідності між графом автомата та його функціями переходів і виходів випливає, що не кожен граф, стрілки якого відмічені парами символів із деяких алфавітів X і Y, буде графом переходів і виходів детермінованого автомата. Для того, щоб відмічений граф був графом детермінованого автомата, необхідно і достатньо, щоб виконувались дві умови, які називаються умовами *автоматності графа*:

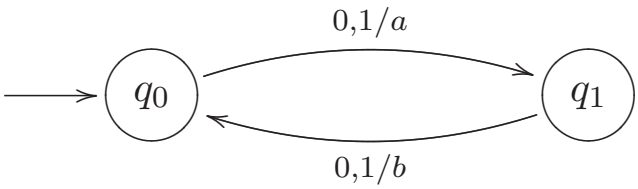
1)не існує двох стрілок з однаковими вхідними відмітками, що виходять з однієї і тієї ж вершини (умова однозначності);

2)для будь-якої вершини і вхідного символу існує стрілка, що виходить із цієї вершини і відмічена цим вхідним символом (умова повноти).

Отже, для недетермінованих автоматів не виконується умова 1), а для часткових – умова 2). Незважаючи на це, як недетерміновані, так і часткові автомати теж зображують графами.

1.3. Приклад. Побудувати детермінований скінченний автомат Мура, який всі букви бінарного слова $\omega \in \{0,1\}^*$, що стоять на непарних позиціях, замінює на букву a, а на парних – на букву b.

Граф переходів і виходів шуканого автомата A має вигляд:



Перебуваючи в стані q_0, даний автомат замінює кожну букву вхідного слова на букву a, а в стані q_1 – на букву b.

Таким чином, шуканий автомат A задається як шістка

$$A = (Q, X, Y, \delta, f, q_0),$$

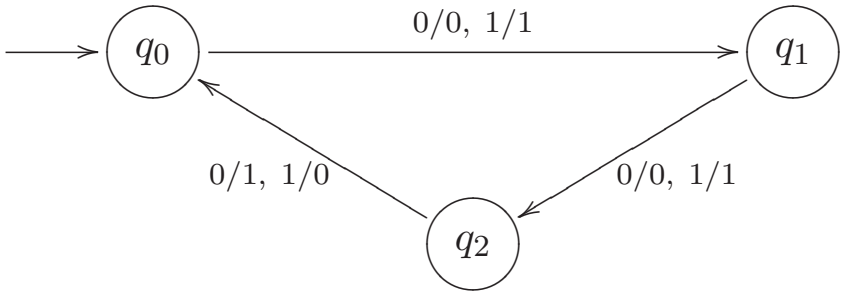
де $Q = \{q_0, q_1\}$, $X = \{0,1\}$, $Y = \{a,b\}$, а функції переходів δ і виходів f задані таблицями:

δ	0	1
q_0	q_1	q_1
q_1	q_0	q_0

f	0	1
q_0	a	a
q_1	b	b

1.4. Приклад. Нехай $X = Y = \{0,1\}$. Побудуємо детермінований скінченний автомат, який „шифрує” всі бітові рядки з 0 та 1, замінюючи всі біти, які стоять на позиціях кратних 3, на протилежні.

Граф переходів і виходів шуканого автомата A має вигляд:



Зауважимо, що якщо „зашифровану" послідовність (вихідне слово) з 0 та 1 подати на вхід цього автомата, то одержимо початкову послідовність (вхідне слово), тобто цей автомат одночасно буде й „дешифруючим пристроєм".

Функції переходів і виходів можна продовжити на вхідну і вихідну напівгрупи $F(X)$ і $F(Y)$. Для цього необхідно покласти

$$\delta(q, e) = q, \delta(q, a\omega) = \delta(\delta(q, a), \omega),$$
$$f(q, e) = e, f(q, a\omega) = f(q, a)f(\delta(q, a), \omega).$$

Використовуючи індукцію за довжиною слова ω, неважко довести таке співвідношення:

$$\delta(q, a\omega) = \delta(\delta(q, a), \omega),$$
$$f(q, a\omega) = f(q, a)f(\delta(q, a), \omega).$$

Дійсно, якщо $\omega = x$, тобто $l(\omega) = 1$, то база індукції має місце за означенням.

Якщо ж $\omega = \nu x$ і для ν співвідношення асоціативності виконується, то

$$\delta(q, a\omega) = \delta(\delta(q, a\nu), x) = \delta(\delta(q, a), \nu x),$$

$$f(q, a\nu x) = f(q, a\omega)f(\delta(q, a\nu), x) =$$

$$= f(q, a)f(\delta(q, a), \alpha)f(\delta(q, a\nu), x) = f(q, a)f(\delta(q, a), \nu x).$$

Якщо $\delta(p, \omega) = q$, то кажемо, що *автомат A переходить із стану p в стан q під дією слова ω*, або, що *слово ω переводить автомат A із стану p в стан q*.

Стан q називається *досяжним* із стану p ($p \to q$), коли існує таке вхідне слово ω, яке переводить автомат A із стану p в стан q. Для того, щоб встановити досяжність q з p, достатньо в графі переходів і виходів знайти шлях із вершини p у вершину q. Очевидно, що коли такий шлях існує, то послідовність перших компонентів відміток стрілок цього шляху і буде складати слово ω.

Зауважимо, що для встановлення досяжності q з p достатньо розглядати лише такі шляхи в графі автомата (вони називаються *простими*), які не проходять двічі через одну і ту ж вершину графа. Це зауваження дає можливість обмежити пошук необхідного вхідного слова лише такими словами, довжина яких не перевищує загальної кількості станів автомата. Ясно, що у випадку скінченних автоматів перевірка істинності відношення $p \to q$ завжди можлива.

Рекомендована література : [**4**, с. 494–509, 552–564], [**5**, с. 385–398], [**11**, с. 286–309], [**12**, с. 410–426], [**16**, с. 285–289].

Питання та вправи до параграфа 1.

1.1. Дайте означення скінченного автомата.

1.2. Які типи автоматів вам відомі?

1.3. В чому полягає різниця між автоматом Мура та автоматом Мілі?

1.4. Які способи задання автоматів ви знаєте?

1.5. Коли ми кажемо, що стан q скінченного автомата A є досяжним із стану p?

1.6. Побудувати граф скінченного автомата $A = (Q, X, Y, \delta, f, q_0)$, де $Q = \{q_0, q_1, q_2\}$, $X = \{0, 1, 2\}$, $Y = \{a, b, c\}$, а функції переходів δ і виходів f задані таблицями:

δ	0	1	2
q_0	q_1	q_1	q_2
q_1	q_1	q_2	q_0
q_2	q_1	q_0	q_2

f	0	1	2
q_0	a	b	a
q_1	b	c	b
q_2	c	a	b

Визначте, в яке слово перетворює автомат A кожне з наступних вхідних слів:

а) 012;
б) 001022;
в) 100000;
г) 000111222.

1.7. Побудувати граф скінченного автомата $A = (Q, X, Y, \delta, f, q_0)$, де $Q = \{q_0, q_1\}$, $X = \{00, 01, 10, 11\}$, $Y = \{0, 1\}$, а функції переходів δ і виходів f задані таблицями:

δ	00	01	10	11
q_0	q_0	q_0	q_0	q_1
q_1	q_0	q_1	q_1	q_1

f	00	01	10	11
q_0	0	1	1	0
q_1	1	0	0	1

В чому полягає робота даного автомата?

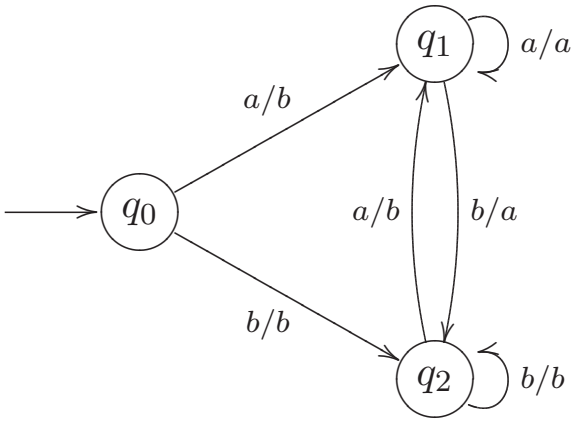
1.8. В чому полягає робота автомата $A = (Q, X, Y, \delta, f, q_0)$, де $Q = \{q_0, q_1\}$, $X = \{0, 1, *\}$, $Y = \{0, 1, +, -\}$, а функції переходів δ і виходів f задані таблицями:

δ	0	1	$*$
q_0	q_0	q_1	q_0
q_1	q_1	q_0	q_0

f	0	1	$*$
q_0	0	1	$+$
q_1	0	1	$-$

?

1.9. Скінченний автомат A заданий графом:



Задати його як шістку $A = (Q, X, Y, \delta, f, q_0)$ і з'ясувати у чому полягає робота автомата.

1.10. Нехай $X = \{a, b, c\}$, $Y = \{*, +, \diamond\}$. Побудувати детермінований скінченний автомат, який реалізує алгоритм шифрування вхідного слова шифром простої заміни, міняючи букву a на $*$, b на $+$ і c на $\diamond$.

1.11. Нехай $X = Y = \{a, b, c, d\}$. Побудувати скінченний автомат, який шифрує букви вхідного слова, які знаходяться на парних позиціях, шифром зсуву на дві букви праворуч, а на непарних – на три букви ліворуч.

1.12. Нехай $X = Y = \{0, 1, 2\}$. Побудувати скінченний автомат, який шифрує вхідне слово, замінюючи цифру x на позиції $3n + k \geq 1$, де $n \geq 0$, $k \in \{0, 1, 2\}$, остачею від ділення $x + k$ на 3.

1.13. Побудувати „дешифруючі автомати" для автоматів з двох попередніх прикладів.

1.14. Побудувати скінченний автомат з виходом, у якого вхідний алфавіт $X = \{0, 1, a\}$ і вихідний $Y = \{0, 1, p, n\}$. Вихідна послідовність з 0 та 1 збігається з вхідною, а на кожний символ запиту a друкується p, якщо кількість 0 від початку роботи парна, і n – якщо непарна.

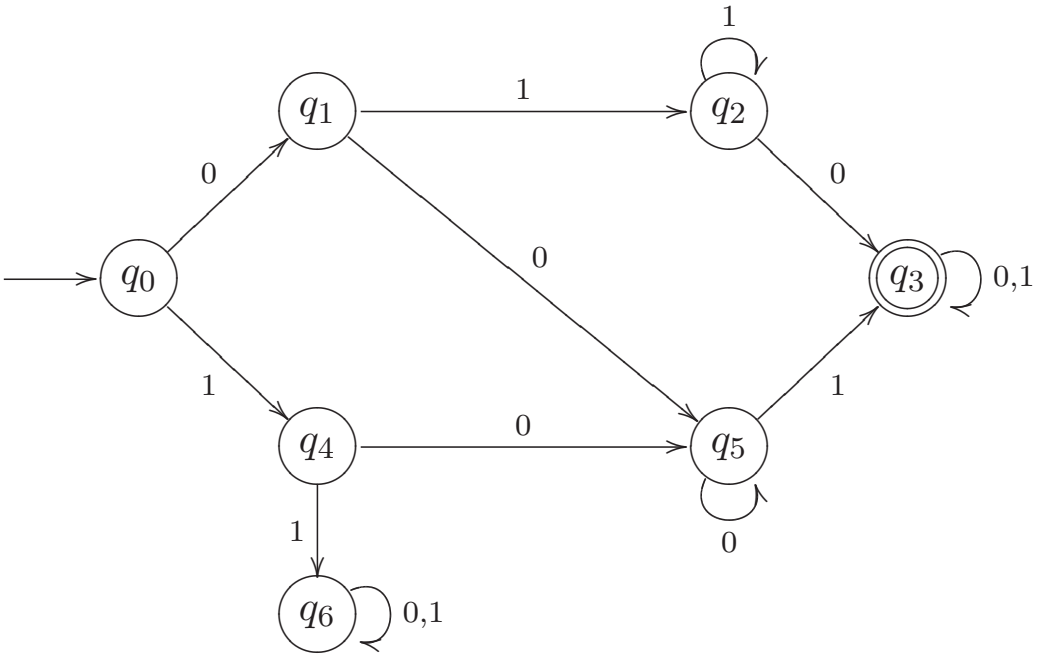
1.15. Нехай $X = Y = \{a, b\}$. Побудувати скінченний автомат, який дає на виході слово, що складається тільки з букв a, тоді і лише тоді, коли вхідне слово містить префікс ab.

1.16. Нехай $X = Y = \{a, b\}$. Побудувати скінченний автомат, який дає на виході слово, що закінчується буквою b, тоді і лише тоді, коли вхідне слово не містить суфікса bbb.

§ 2. Детерміновані скінченні автомати без виходу

В цьому параграфі ми на прикладах розглянемо методи побудови детермінованих скінченних автоматів (ДСА) без виходу. Зокрема покажемо, що клас формальних мов, які розпізнаються детермінованими скінченними автоматами, є замкненим відносно об'єднання, перетину, доповнення, різниці та симетричної різниці.

2.1. Приклад. Знайти формальну мову, що розпізнається ДСА $A = (Q, X, \delta, q_0, F)$, який заданий поміченим графом:



Нагадаємо, що слово $\omega \in L(A)$ тоді і тільки тоді, коли автомат A зупиняє аналіз ω в одному з кінцевих станів множини F. Якщо при аналізі слова ω автомат переходить в стан q_6, то дане слово не розпізнається A, і навпаки, перейшовши в кінцевий стан q_3 автомат розпізнає слово ω. Отже, нам потрібно проаналізувати, як зі стану q_0 можна перейти в стан q_3. Всього є три типи шляхів з q_0 в q_3:

$$(q_0, q_1, q_2, \ldots, q_2, q_3); \quad (q_0, q_1, q_5, \ldots, q_5, q_3); \quad (q_0, q_4, q_5, \ldots, q_5, q_3).$$

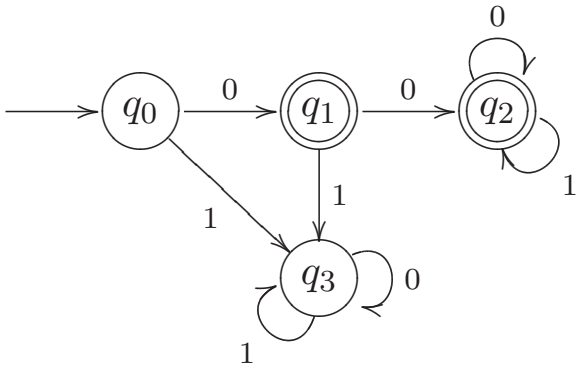
Вони відповідають словам, префікси яких належать регулярним мовам 011^*0, 000^*1 і 100^*1 відповідно. Таким чином,

$$L(A) = (011^*0 + 000^*1 + 100^*1)(0 + 1)^*.$$

2.2. Приклад. Побудувати граф ДСА $A = (Q, X, \delta, q_0, F)$ і знайти мову, яка розпізнається A, якщо $Q = \{q_0, q_1, q_2, q_3\}$, $X = \{0, 1\}$, $F = \{q_1, q_2\}$, а функція переходів δ задана таблицею:

δ	0	1
q_0	q_1	q_3
q_1	q_2	q_3
q_2	q_2	q_2
q_3	q_3	q_3

Помічений граф автомата A має вигляд:

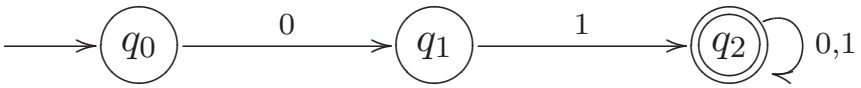


Знайдемо мову $L(A)$, яка розпізнається автоматом A.

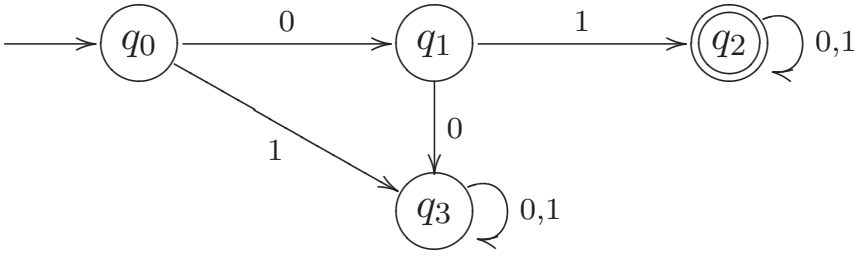
З графа ДСА видно: перейшовши в стан q_3, автомат залишається в ньому до кінця роботи і $\omega \notin L(A)$. З іншого боку, перейшовши в кінцевий стан q_2, автомат ніколи не покине даного стану, і такі слова розпізнаються ДСА. З вищесказаного випливає, що мова $L(A)$ містить слово 0 (аналіз якого зупиняється в стані q_1) і всі слова, які мають префікс 00. Таким чином, $L(A)$ записується регулярним виразом $0 + 00(0 + 1)^*$.

2.3. Приклад. Побудувати ДСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, які мають префікс 01.

Легко бачити, що регулярний вираз шуканої формальної мови має вигляд $01(0 + 1)^*$. Побудуємо граф цього регулярного виразу:



Даний граф не є графом ДСА, оскільки в графі ДСА з кожної вершини q має виходити єдина стрілка з позначкою a для кожної пари $(q, a) \in Q \times X$. Щоб виконати цю умову, побудуємо додаткову тупикову вершину q_3 і стрілки $q_0 \to q_3$ і $q_1 \to q_3$ з позначками 1 і 0 відповідно. Шуканий граф ДСА має вигляд:



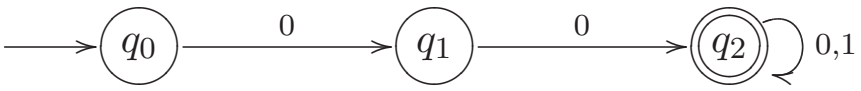
2.4. Приклад. Побудувати ДСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, які містять підслово 00.

Спершу зауважимо, що з регулярного виразу $(0 + 1)^*00(0 + 1)^*$ не можна визначити перше входження пари 00 у вхідному слові.

З другого боку, шуканий ДСА має виявляти 00 вже при першому його входженні. Проаналізуємо дану проблему детальніше.

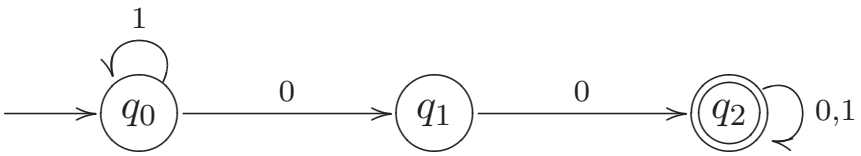
Припустимо, що на стрічці записане слово $\omega = x_1x_2\dots x_n$, де $x_i \in \{0,1\}$. Ми маємо перевіряти кожне підслово $x_1x_2, x_2x_3, \dots, x_{n-1}x_n$, і як тільки виявиться, що деяка пара рівна 00, робити висновок, що слово w розпізнається ДСА. Звідси випливає наступний алгоритм побудови графа автомата:

Крок 1. Будуємо граф

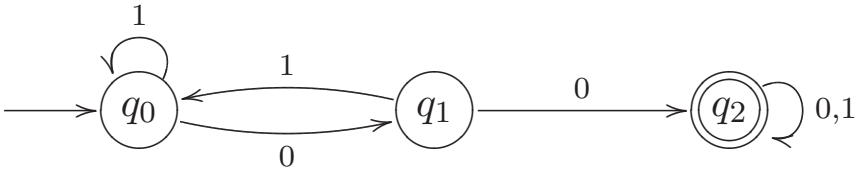


з кінцевою вершиною q_2, з допомогою якої розпізнаються слова, що містять 00. Зокрема, якщо $x_1x_2 = 00$, то слово ω розпізнається.

Крок 2. Якщо $x_1 = 1$, то залишаємо підслово x_1x_2 і переходимо до перевірки x_2x_3. Таким чином, автомату потрібно перейти в стан q_0, тобто $\delta(q_0, 1) = q_0$.

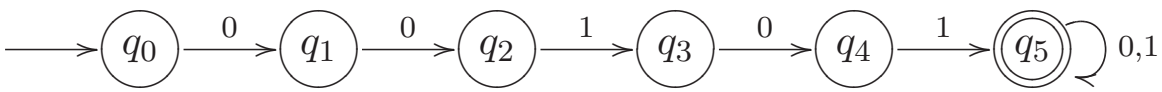


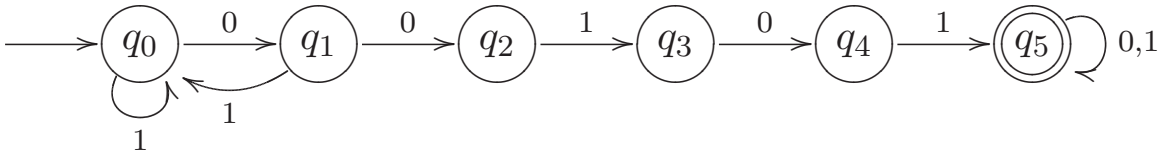
Крок 3. Якщо $x_1 = 0$ і $x_2 = 1$, то ні підслово x_1x_2, ні x_2x_3 не дорівнює 00. Отже, керуючий пристрій ДСА має перейти в стан q_0 для аналізу підслова x_3x_4. Таким чином, покладемо $\delta(q_1, 1) = q_0$. Граф шуканого ДСА має вигляд:

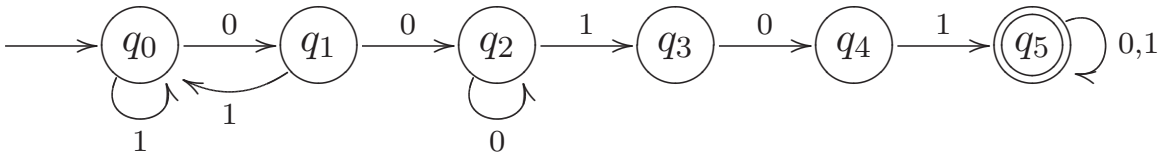


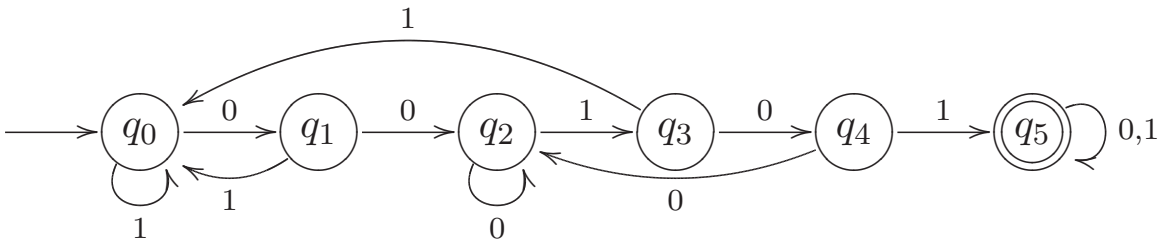
2.5. Приклад. Побудувати ДСА, який розпізнає всі слова в бінарному алфавіті $X = \{0,1\}$, які містять підслово 00101.

Аналогічно, як і в попередньому прикладі, крок за кроком будуємо граф ДСА:



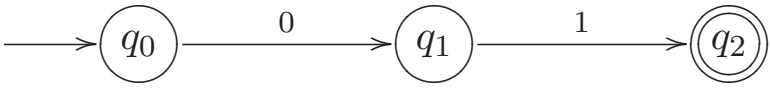




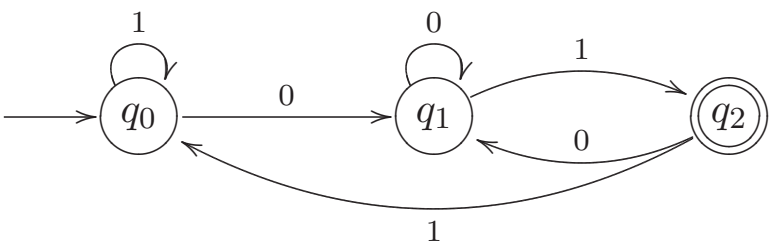


2.6. Приклад. Побудувати ДСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, які мають суфікс 01.

Спершу побудуємо наступний граф:



Стани q_0 та q_1 вказують, що „не знайдено префікса слова 01” та „знайдений префікс 0 слова 01” відповідно. Аналогічно, як і в попередньому прикладі, покладемо $\delta(q_0, 1) = q_0$ і $\delta(q_1, 0) = q_1$. Якщо перейшовши в стан q_2 автомат не закінчив аналіз вхідного слова, то необхідно покласти $\delta(q_2, 0) = q_1$ і $\delta(q_2, 1) = q_0$. Таким чином, шуканий граф ДСА має вигляд:



2.7. Приклад. Побудувати ДСА, який розпізнає всі двійкові натуральні числа, які конгруентні нулю за модулем 5.

Ідея побудови даного ДСА подібна до ідеї побудови автоматів з попередніх двох прикладів. Побудуємо п'ять станів $q_0, q_1, \ldots, q_4$ і вважаємо, що кожен стан q_i має значення „префікс α вхідного слова має властивість $\alpha \equiv i(mod\ 5)$". Тобто потрібно визначити $\delta(q_0, x_1x_2 \ldots x_k) = q_i$, якщо $x_1x_2 \ldots x_k \equiv i(mod\ 5)$.

Як же побудувати стрілки між станами автомата, використовуючи цю ідею? Нагадаємо, що для функції переходів δ виконується наступна рівність:

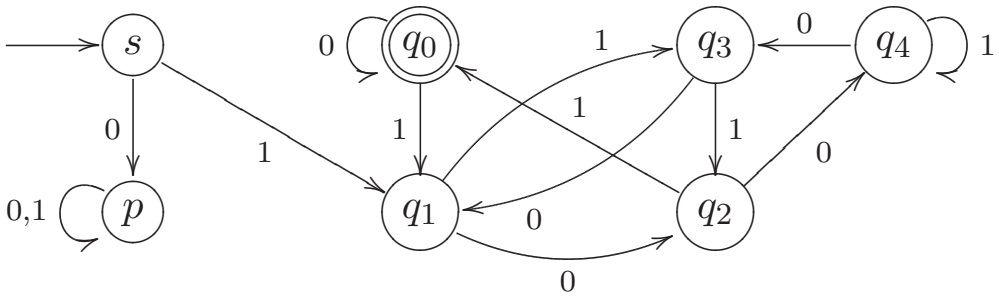
$$\delta(\delta(q_0, \omega), a) = \delta(q_0, \omega a)$$

для будь-якого бінарного слова ω і довільного $a \in \{0, 1\}$. Припустимо, що $\delta(q_0, \omega) = q_i$ і $\delta(q_0, \omega a) = q_j$. Тоді мають виконуватися конгруенції: $\omega \equiv i(mod\ 5)$ і $\omega a \equiv j(mod\ 5)$. Таким чином,

$$j \equiv \omega a(mod\ 5) \equiv 2 \cdot \omega + a(mod\ 5) \equiv 2 \cdot i + a(mod\ 5).$$

Отже, покладемо $\delta(q_i, a) = q_j$, якщо $j \equiv 2 \cdot i + a(mod\ 5)$. Наприклад, $\delta(q_2, 0) = q_4$ і $\delta(q_2, 1) = q_0$. Крім того, стан q_0 є єдиним кінцевим станом, оскільки $\delta(q_0, \omega) = q_0$ означає, що $\omega \equiv 0(mod\ 5)$.

Насамкінець зауважимо, що двійковий запис натурального числа завжди починається з 1. Таким чином, потрібно додати новий початковий стан s і тупиковий стан p як показано на графі:



2.8. Приклад. Побудувати ДСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, які містять підслово 00 або закінчуються на 01.

Шукана мова є об'єднанням двох мов, записаних регулярними виразами $(0+1)^*00(0+1)^*$ і $(0+1)^*01$. В прикладах 2.4 та 2.6 ми побудували ДСА, які розпізнають ці дві мови. Для перевірки того, чи вхідне слово ω належить об'єднанню цих мов, можна аналізувати слово ω двома автоматами паралельно. Наприклад, нехай $\omega = 0101$.

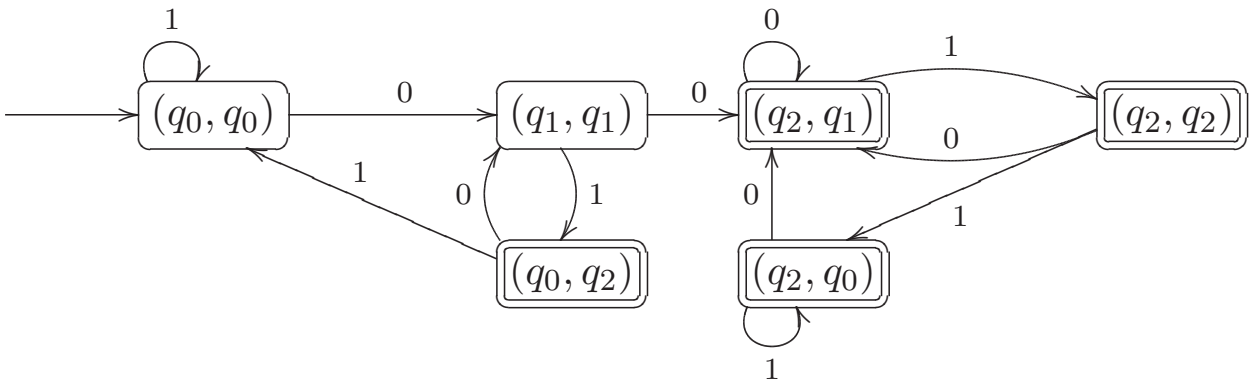
На першому ДСА обчислювальний шлях для слова ω має вигляд $(q_0, q_1, q_0, q_1, q_0)$, а на другому – $(q_0, q_1, q_2, q_1, q_2)$. Оскільки аналіз ω на другому ДСА закінчується в кінцевому стані, то слово належить об'єднанню мов.

Ідея побудови ДСА для об'єднання цих мов полягає в розгляді так званого добутку автоматів. Нехай $A_1 = (Q_1, X, \delta_1, q_0, F_1)$ і $A_2 = (Q_2, X, \delta_2, q_0, F_2)$ – два автомати (зауважимо, що Q_1 і Q_2 можуть мати стани з однаковими назвами, які виконують різні функції в двох ДСА). Визначимо *добуток $A = A_1 \times A_2$ автоматів A_1 і A_2* наступним чином. Нехай $A = (Q, X, \delta, \widetilde{q}_0, F)$. Покладемо

$$Q = Q_1 \times Q_2 = \{(q_i, q_j) \mid q_i \in Q_1, q_j \in Q_2\},$$

$$\delta((q_i, q_j), a) = (\delta_1(q_i, a), \delta_2(q_j, a)) \quad \text{і} \quad \widetilde{q}_0 = (q_0, q_0).$$

Наприклад, обчислювальний шлях слова $\omega = 0101$ в добутку автоматів A має вигляд: (q_0, q_0); (q_1, q_1); (q_0, q_2); (q_1, q_1); (q_0, q_2). Якщо $q_i \in F_1$ або $q_j \in F_2$, то покладаємо $(q_i, q_j) \in F$, тобто $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$. Очевидно, що даний автомат A розпізнає об'єднання мов, які розпізнаються автоматами A_1 та A_2. Граф шуканого ДСА має вигляд:



Звернемо увагу на два важливі факти, пов'язані з автоматом A. По-перше, оскільки стани (q_0, q_1), (q_1, q_0) та (q_1, q_2) є недосяжними з початкового стану (q_0, q_0), то ми їх опускаємо. По-друге, стани (q_2, q_1), (q_2, q_0) та (q_2, q_2) можуть бути об'єднані в єдиний стан, оскільки всі вони є кінцевими і неможливо їх покинути, якщо автомат перейшов хоча б в один із них.

2.9. Приклад. Побудувати ДСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, які містять підслово 00 і закінчуються на 01.

Дана мова є перетином двох мов, записаних регулярними виразами $(0+1)^*00(0+1)^*$ і $(0+1)^*01$. В прикладі 2.8 побудовано добуток автоматів, який розпізнає об'єднання заданих мов. Тут ми розглянемо той самий добуток ДСА, в якому лише поміняємо множину кінцевих станів, поклавши $F = F_1 \times F_2$. Граф шуканого автомата такий же, як і автомата з прикладу 2.8, з тією різницею, що множина F кінцевих станів містить тільки стан (q_2, q_2).

2.10. Приклад. Побудувати ДСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, які містять підслово 00 і не закінчуються на 01.

Дана мова є різницею мов, записаних регулярними виразами $(0 + 1)^*00(0 + 1)^*$ і $(0 + 1)^*01$. Тому ми можемо використати той самий добуток автоматів, що і в прикладах 2.8 та 2.9, включивши в множину кінцевих станів ті пари, перша компонента яких є кінцевим станом першого ДСА, а друга компонента не є кінцевим станом другого автомата, тобто поклавши $F = F_1 \times (Q_2 \setminus F_2)$. Граф шуканого автомата такий же, як і автомата з прикладу 2.8, з тією різницею, що множина F кінцевих станів містить стани (q_2, q_0) і (q_2, q_1).

Частковим випадком різниці формальних мов є доповнення $\overline{L} = X^* \setminus L$. В цьому випадку використовуємо наступну конструкцію. Зауважимо, що для автомата $A = (Q, X, \delta, q_0, F)$ вхідне слово $\omega \in L(A)$ тоді і тільки тоді, коли $\delta(q_0, \omega) \in F$. Аналогічно $\omega \notin L(A)$ тоді і лише тоді, коли $\delta(q_0, \omega) \notin F$. Звідси випливає, що ДСА $(Q, X, \delta, q_0, Q \setminus F)$ розпізнає доповнення мови $L(A)$.

Таким чином, ми довели наступну теорему:

2.11. Теорема. *Клас формальних мов, які розпізнаються детермінованими скінченними автоматами, є замкненим відносно скінченних об'єднань, перетинів, доповнення, різниці та симетричної різниці.*

Рекомендована література : [**15**, с. 33–45], [**16**, с. 289–294], [**22**, с. 23–38].

Питання та вправи до параграфа 2.

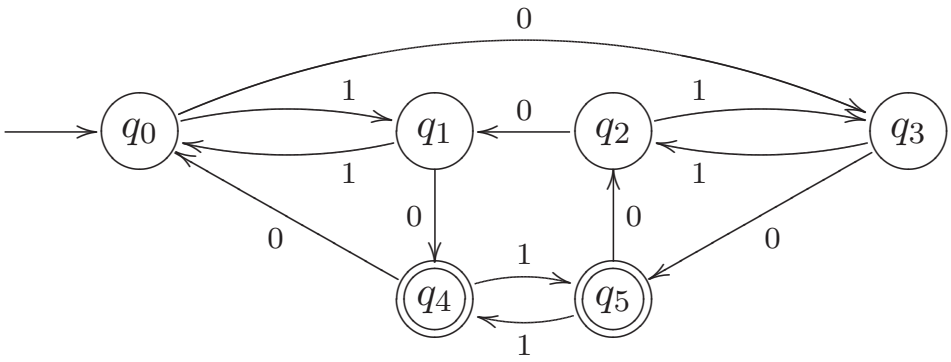
2.1. Як будується добуток $A = A_1 \times A_2$ автоматів A_1 і A_2?

2.2. Сформулюйте теорему про замкненість класу формальних мов, які розпізнаються детермінованими скінченними автоматами.

2.3. Визначити, який з бітових рядків

а) 1110
б) 101010
в) 110001010010
г) 000000000111

розпізнається детермінованим скінченним автоматом без виходу A, що заданий графом:



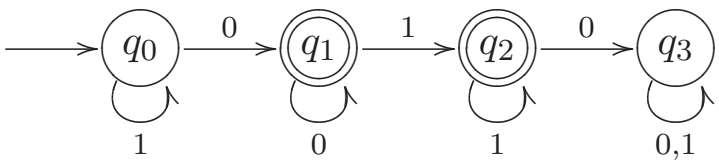
2.4. Серед слів регулярної мови $(10)^*$ визначити ті, які належать мові $L(A)$ автомата A з попереднього прикладу.

2.5. Розглянемо ДСА без виходу $A_{m,d} = (Q, X, \delta, q_0, F)$, де $m, d \in \mathbb{N}$, $Q = \{q_0, q_1, \ldots, q_{m-1}\}$, $X = \{0, 1, \ldots, d-1\}$, $F = \{q_1\}$ і $\delta(q_i, k) = q_{(di+k) mod\ m}$. Побудувати граф автомата $A_{7,2}$ і визначити, який з бітових рядків розпізнається ДСА:

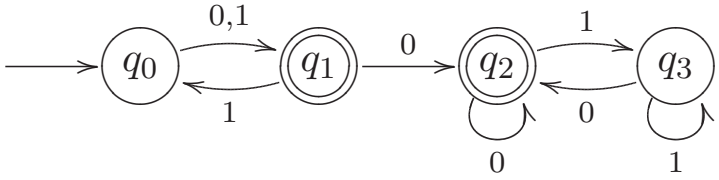
а) 0101
б) 11010
в) 1101101010
г) 11110000.

2.6. Знайти мову, що розпізнається ДСА без виходу:

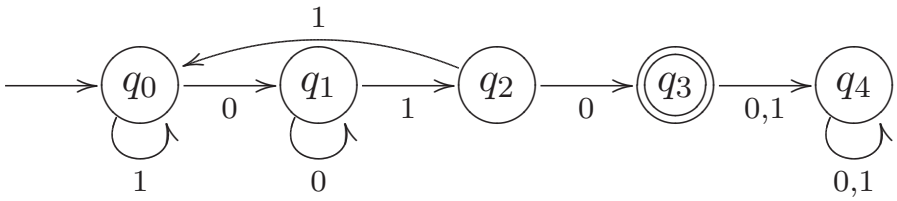
а)



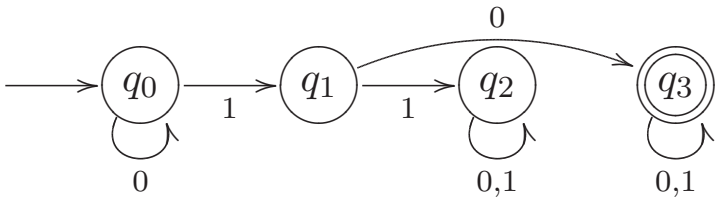
б)



в)



г)



2.7. Побудувати ДСА без виходу, які розпізнають формальні мови:

а) $\{0, 1\}$;
б) $\{10, 101\}$;
в) $\{0^n \mid n = 2, 3, \ldots\}$;
г) $\{0^n 1^m \mid n, m \in \mathbb{N}\}$.

2.8. Побудувати ДСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, що містять підслово 100101.

2.9. Побудувати ДСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, що мають префікс 010.

2.10. Побудувати ДСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, що закінчуються на 101.

2.11. Побудувати ДСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, що починаються на 010 або закінчуються на 101.

2.12. Побудувати ДСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, що містять підслово 11 і не закінчуються на 101.

2.13. Побудувати ДСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, що мають префікс 010, суфікс 101 і містять підслово 0000.

§ 3. Недетерміновані скінченні автомати без виходу

Недетермінований скінченний автомат (НСА) без виходу $A = (Q, X, \delta, q_0, F)$ визначається так само, як і ДСА, за винятком того, що δ не обов'язково є всюди визначеною функцією, а може бути й відношенням. Це означає, що для кожного стану q і вхідної букви a значення $\delta(q, a)$ є підмножиною множини станів Q, $\delta(q, a) = \{p_1, p_2, \ldots, p_k\}$, тобто проаналізувавши букву a в стані q автомат може перейти в довільний зі станів p_1, $p_2, \ldots$, p_k. Якщо $\delta(q, a) = \emptyset$, то автомат не переходить в жоден стан, і вважається, що вхідне слово не розпізнається НСА, незважаючи на те що деякі букви слова ще не проаналізовані автоматом. В цьому випадку кажуть, що НСА перейшов в тупиковий стан. Крім переходів в декілька станів, в НСА дозволяються також e-переходи (e-такти). При e-переході головка автомата нічого не виконує (не читає і не рухається), але стан при цьому може змінитися на довільний із заданих цим переходом станів.

З сказаного вище випливає, що δ можна визначити як функцію, значення якої є підмножинами множини станів Q, тобто

$$\delta : Q \times (X \cup \{e\}) \to 2^Q,$$

де через 2^Q позначається сім'я всіх підмножин множини Q.

Функцію δ зручно задавати за допомогою таблиці, стовпчики якої відмічені різними буквами вхідного алфавіту, а рядки – станами автомата. Для функції переходів на перетині i-го рядка, відміченого станом $q \in Q$, і j-го стовпчика, відміченого буквою $a \in X \cup \{e\}$, знаходиться значення $\delta(q, a) = \{p_1, p_2, \ldots, p_k\}$.

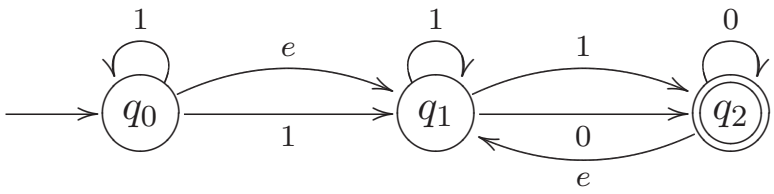
3.1. Приклад. Функція δ автомата

$$A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$$

задається таблицею:

δ	0	1	e
q_0	$\emptyset$	$\{q_0, q_1\}$	$\{q_1\}$
q_1	$\{q_2\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_2\}$	$\emptyset$	$\{q_1\}$

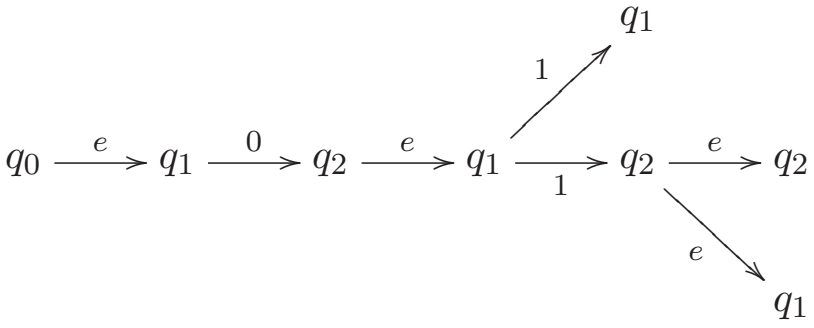
НСА, так як і ДСА, задаються графами, вершинами яких є стани автомата, а за допомогою стрілок зображуються переходи. Якщо $\delta(q,a)=\{p_1,p_2,\dots,p_k\}$, то проводиться k стрілок з вершини q до кожної з вершин $p_1,p_2,\dots,p_k$, і всі стрілки помічаються буквою a. Наприклад, граф автомата A з прикладу 3.1 має вигляд:



Вхідне слово ω може мати більше одного обчислювального шляху. Наприклад, слово $\omega=01$ має три обчислювальні шляхи:

$$q_0 \xrightarrow{e} q_1 \xrightarrow{0} q_2 \xrightarrow{e} q_1 \xrightarrow{1} q_1,$$
$$q_0 \xrightarrow{e} q_1 \xrightarrow{0} q_2 \xrightarrow{e} q_1 \xrightarrow{1} q_2,$$
$$q_0 \xrightarrow{e} q_1 \xrightarrow{0} q_2 \xrightarrow{e} q_1 \xrightarrow{1} q_2 \xrightarrow{e} q_1.$$

Обчислювальні шляхи вхідного слова ω утворюють кореневе дерево виведення, бо всі вони виходять з однієї і тієї ж вершини q_0 та їх гілки не утворюють циклів. Зобразимо дерево виведення слова $\omega=01$ (ми додаємо стрілку $q_2 \xrightarrow{e} q_2$, щоб наголосити, що другий шлях закінчується в вершині q_2):



Деякі з цих обчислювальних шляхів закінчуються в кінцевому стані, а інші – ні. Як же в цьому випадку визначити, чи розпізнається вхідне слово НСА? Вважають, що автомат розпізнає слово ω, якщо принаймні один з обчислювальних шляхів закінчується в кінцевому стані. Наприклад, другий обчислювальний шлях слова $\omega=01$ закінчується в q_2, і тому воно розпізнається НСА A з прикладу 3.1.

Щоб визначити строго, коли НСА розпізнає вхідне слово ω, потрібно ввести поняття e-замикання множини. Назвемо *e-замиканням* підмножини $P \subset Q$ множину станів, які досягаються з станів $q \in P$ e-переходами (включаючи переходи з q в q). Тобто

$$cl_e(P) = \{p \in Q | (p \in P) \vee (\exists q_0, \ldots, q_m)[q_0 \in P, q_m = p, q_{i+1} \in \delta(q_i, e)]\}.$$

Наприклад, $cl_e(\{q_0\}) = \{q_0, q_1\}$, а $cl_e(\{q_2\}) = \{q_1, q_2\}$ для автомата з прикладу 3.1.

Продовжимо функцію переходів δ на множину $2^Q \times (X \cup \{e\})$, поклавши

$$\delta(P, a) = cl_e(\bigcup_{q \in cl_e(P)} \delta(q, a)),$$

а далі продовжимо її на $2^Q \times X^*$ наступним чином:

$$\delta(P, e) = cl_e(P),$$

$$\delta(P, \omega a) = \delta(\delta(P, \omega), a), \text{ якщо } \omega \in X^* \text{ і } a \in X.$$

Так, у прикладі 3.1 маємо, що $\delta(\{q_0\}, 0) = \{q_1, q_2\}$, а $\delta(\{q_1, q_2\}, 1) = \{q_1, q_2\}$, тому $\delta(\{q_0\}, 01) = \{q_1, q_2\}$.

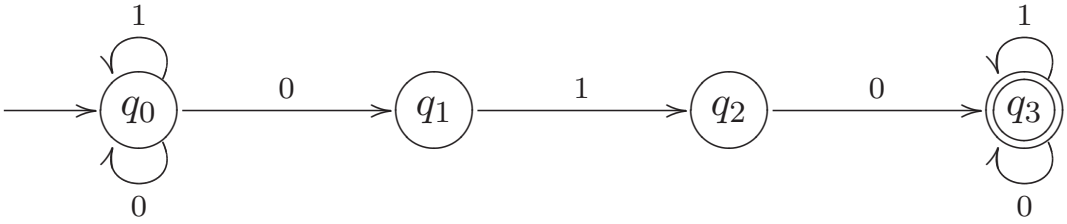
Зауважимо, що $\delta(\{q_0\}, \omega)$ є множиною всіх останніх станів (не обов'язково кінцевих!) обчислювальних шляхів слова ω.

Тепер можна строго визначити, що *НСА (Q, X, δ, q_0, F) розпізнає слово ω*, якщо $\delta(\{q_0\}, \omega) \cap F \neq \emptyset$. Через $L(A)$ позначаємо мову, яка розпізнається НСА A, тобто

$$L(A) = \{\omega \in X^* \mid \delta(\{q_0\}, \omega) \cap F \neq \emptyset\}.$$

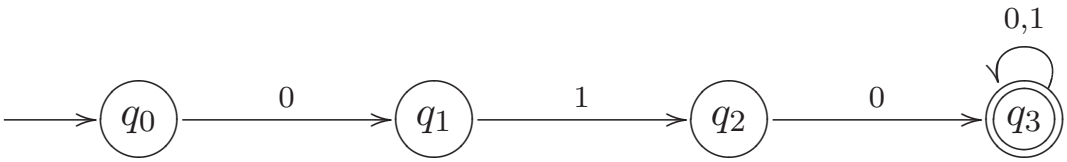
3.2. Приклад. Побудувати НСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, що містять підслово 010.

Будуємо даний автомат так само, як ми будували ДСА, тільки в початковому стані додаємо дві петлі з позначками 0 і 1. З їх допомогою автомат чекає доти, доки не знайде підслово 010. Маючи ці дві петлі, не потрібно покладати $\delta(q_1, 0) = q_1$ і $\delta(q_2, 1) = q_0$, як це робилося для ДСА. Справді, достатньо просто покласти $\delta(q_1, 0) = \delta(q_2, 1) = \emptyset$. Граф НСА має вигляд:

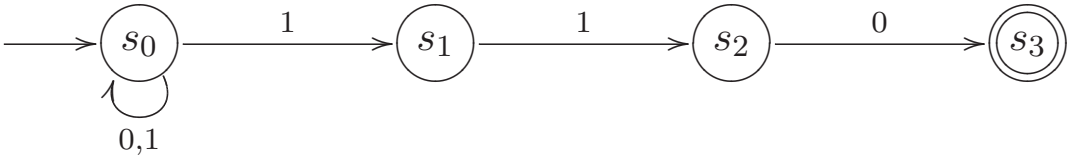


3.3. Приклад. Побудуємо НСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, що починаються з 010 або закінчуються на 110.

Множина всіх бінарних слів, які починаються з 010, розпізнається НСА, заданим графом:

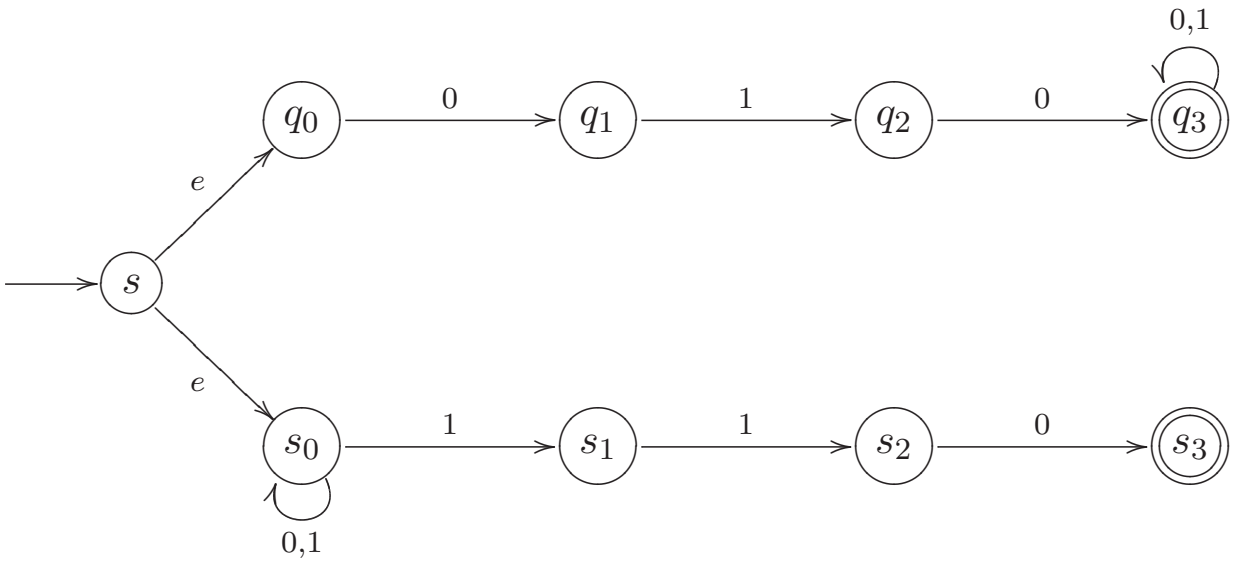


Множина всіх бінарних слів, які закінчуються на 110, розпізнається автоматом:



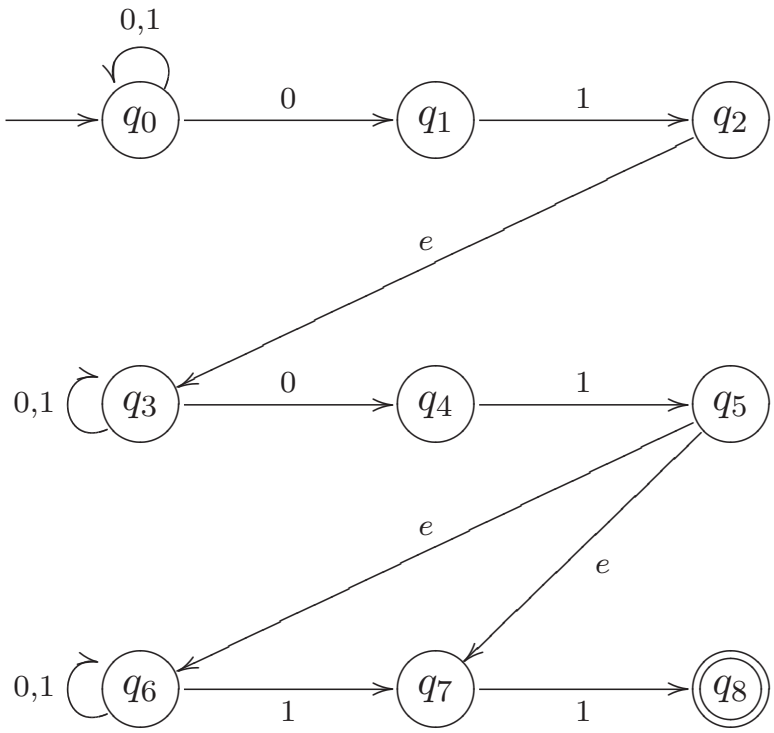
Зауважимо, що в цьому автоматі кінцевий стан не має вихідних стрілок, тобто $\delta(s_3, 0) = \delta(s_3, 1) = \emptyset$. Таким чином, якщо аналізуючи вхідне слово ω, автомат переходить в тупиковий кінцевий стан s_3 раніше, ніж слово ω повністю проаналізується НСА, то з допомогою цього шляху ω не розпізнається (але це не означає, що ω не розпізнається НСА).

Насамкінець об'єднаємо ці два графи в один, додавши новий початковий стан і дві e-стрілки, які виходять з нього в старі початкові стани. В результаті одержимо наступний граф шуканого НСА:



3.4. Приклад. Побудуємо НСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, що містять принаймні два входження підслова 01 і закінчуються на 11.

Використаємо e-переходи, щоб об'єднати три простіших НСА в один. В результаті отримаємо:



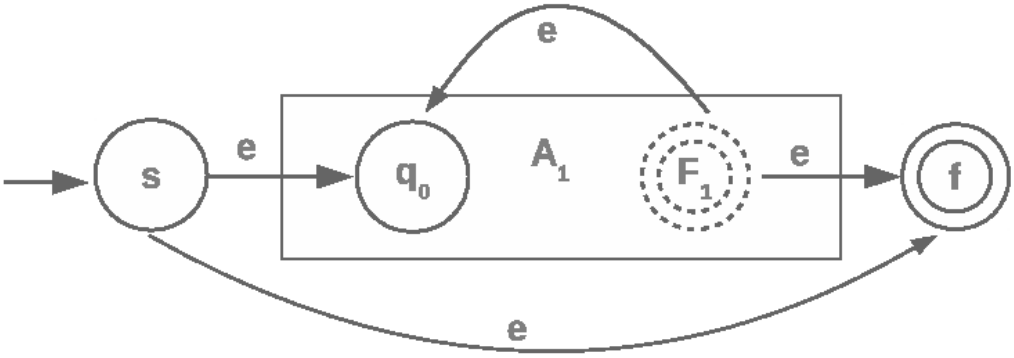
Зауважимо, що буква 1 другого входження підслова 01 може співпасти з першою буквою суфікса 11, але це не є проблемою у випадку НСА: слід просто додати ще одну e-стрілку з стану q_5 в стан q_7.

3.5. Приклад. Нехай A_1 і A_2 – два НСА. Побудуємо НСА A, для якого $L(A) = L(A_1) \circ L(A_2)$.

Нехай $A_1 = (Q_1, X, \delta_1, q_0^1, F_1)$ і $A_2 = (Q_2, X, \delta_2, q_0^2, F_2)$, де $Q_1 \cap Q_2 = \emptyset$. Побудуємо автомат $A = (Q, X, \delta, q_0, F)$ наступним чином. Покладемо $Q = Q_1 \cup Q_2$. Початковий стан q_0^1 автомата A_1 є початковим станом автомата A, а множина F кінцевих станів A дорівнює F_2. Також для кожного $q \in F_1$ довизначимо $\delta(q, e) = q_0^2$, тобто проведемо e-стрілки з кожного стану $q \in F_1$ в початковий стан q_0^2 автомата A_2. На решту наборах функція δ визначається так само, як функції δ_1 і δ_2. Легко бачити, що автомат A розпізнає мову $L(A) = L(A_1) \circ L(A_2)$.

3.6. Приклад. Нехай A_1 – НСА. Побудуємо НСА A, для якого $L(A) = L(A_1)^*$.

Нехай $A_1 = (Q_1, X, \delta_1, q_0, F_1)$. Побудуємо граф автомата A, додавши новий початковий стан s і єдиний кінцевий стан f. Проведемо e-стрілки з стану s в старий початковий стан $q_0 \in Q_1$ та з кожного $q_i \in F_1$ в новий кінцевий стан f. Далі відкладемо з кожного стану $q_i \in F_1$ e-стрілку в початковий стан q_0 автомата A_1. Насамкінець додамо e-стрілку з початкового стану s в новий кінцевий стан f (при цьому порожнє слово e розпізнається автоматом). Шуканий НСА показано на рисунку:



З прикладів 3.5 та 3.6 випливає наступна теорема:

3.7. Теорема. *Клас формальних мов, які розпізнаються недетермінованими скінченними автоматами, замкнений відносно конкатенації та ітерації.*

Рекомендована література : [**2**, с. 134–138], [**11**, с. 309–312], [**15**, с. 55–60], [**22**, с. 38–45].

Питання та вправи до параграфа 3.

3.1. У чому полягає різниця між ДСА і НСА?

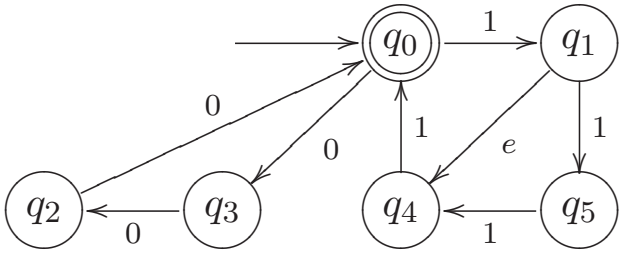
3.2. Як за деревом виведення вхідного слова з'ясувати, чи розпізнається воно автоматом?

3.3. Сформулюйте теорему про замкненість класу формальних мов, які розпізнаються недетермінованими скінченними автоматами.

3.4. Визначити, який з бітових рядків

а) 00010
б) 11000
в) 110001111000
г) 000000000111

розпізнається недетермінованим скінченним автоматом без виходу A, що заданий графом:

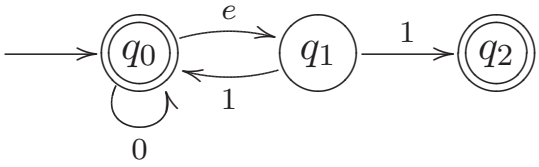


Знайти e-замикання множини $\{q_0, q_1\}$.

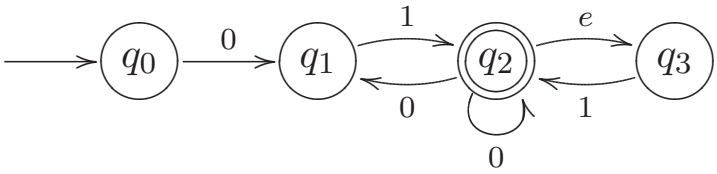
3.5. Серед слів регулярної мови 0^*1^* визначити ті, які належать мові $L(A)$ НСА без виходу A з попереднього прикладу. Задати його як п'ятірку $A = (Q, X, \delta, q_0, F)$.

3.6. Знайти мову, що розпізнається НСА без виходу:

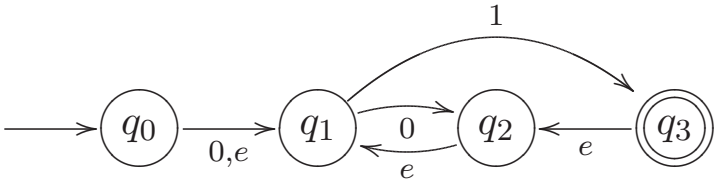
а)



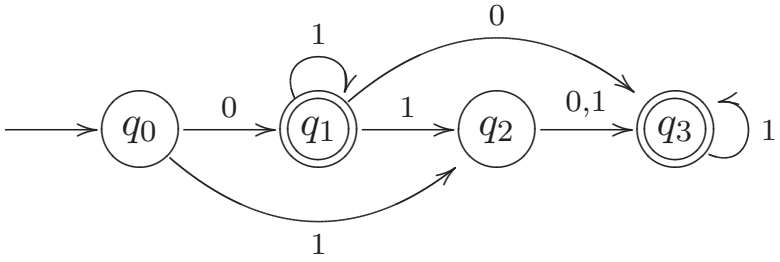
б)



в)



г)



3.7. Побудувати дерева виведення слова 010011 для кожного автомата з попереднього прикладу і з їх допомогою з'ясувати, які з автоматів розпізнають дане слово.

3.8. Побудувати НСА без виходу, які розпізнають формальні мови:

а) $\{1, e\}$;
б) $\{01, 101\}$;
в) $\{0^n \mid n = 2, 3, \ldots\}$;
г) $\{0^n 1^m \mid n, m \in \mathbb{N}\}$.

3.9. Побудувати НСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, що містять підслово 00101.

3.10. Побудувати НСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, що мають префікс 111.

3.11. Побудувати НСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, що закінчуються на 001.

3.12. Побудувати НСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, що починаються на 111 або закінчуються на 001.

3.13. Побудувати НСА, який розпізнає всі слова в алфавіті $X = \{0, 1\}$, що містять щонайменше три входження підслова 010.

3.14. Користуючись розглянутою в прикладі 3.5 параграфа 3 конструкцією побудувати автомат, який розпізнає всі слова в алфавіті $X = \{0, 1\}$, що починаються на 11 і закінчуються на 01.

3.15. Користуючись розглянутою в прикладі 3.6 параграфа 3 конструкцією побудувати НСА, який розпізнає регулярну мову $\{1, 01\}^*$ в алфавіті $X = \{0, 1\}$.

§ 4. Перетворення НСА до ДСА

Недетерміновані скінченні автомати будуються простіше, ніж детерміновані скінченні автомати, але вони є ідеалізованими машинами і не можуть бути ефективно застосовані на практиці, бо реальна машина може працювати тільки за однозначно визначеним алгоритмом. Виявляється, що є простий метод для перетворення НСА в рівносильний ДСА в тому розумінні, що обидва автомати розпізнають одну і ту ж формальну мову.

Розглянемо НСА $A = (Q, X, \delta, q_0, F)$. Для кожного $\omega \in X^*$ позначимо через Q_ω множину останніх (не обов'язково кінцевих!) станів обчислювальних шляхів слова ω, тобто $Q_\omega = \delta(\{q_0\}, \omega)$, де δ – функція переходів, визначена на множині $2^Q \times X^*$. Зокрема, $Q_e = cl_e(\{q_0\})$. Таким чином, слово ω розпізнається НСА тоді і тільки тоді, коли $Q_\omega \cap F \neq \emptyset$. Отже, можна взяти підмножини $Q_\omega \subset Q$ в якості станів нового рівносильного ДСА. Іншими словами, побудуємо ДСА $A' = (Q', X, \delta', Q_e, F')$ з наступними компонентами:

$$Q' = \{Q_\omega \mid \omega \in X^*\}, \quad F' = \{Q_\omega \mid Q_\omega \cap F \neq \emptyset\},$$

$$\delta'(Q_\omega, a) = Q_{\omega a}, \text{ де } \omega \in X^*, a \in X.$$

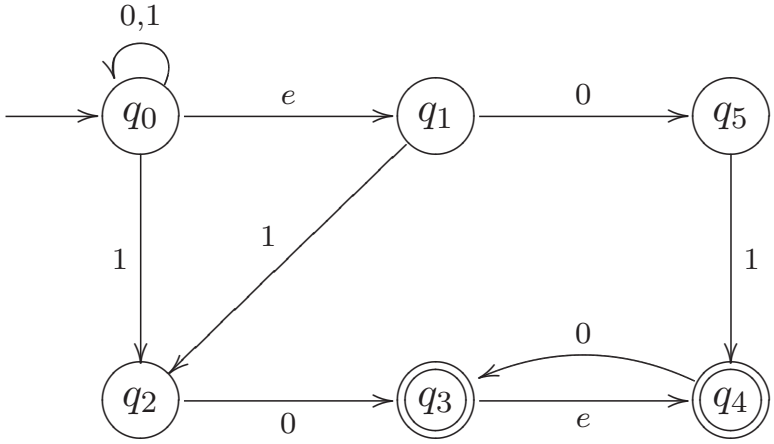
Кожен стан $Q_\omega \in Q'$ є підмножиною множини Q. Оскільки множина Q містить $2^{|Q|}$ підмножин, то Q' є скінченною множиною. Якщо $Q_\omega = Q_v$, то $Q_{\omega a} = Q_{va}$ для всіх $a \in X$. Звідси випливає, що означення функції переходів δ' є коректним. Дана конструкція називається *побудовою ДСА за допомогою підмножин*.

Розглянемо деякі приклади.

4.1. Приклад. Побудувати ДСА, який розпізнає ту ж формальну мову, що і НСА $A = (Q, \{0, 1\}, \delta, q_0, F)$, де $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$, $F = \{q_3, q_4\}$ і функція переходів задана таблицею:

δ	0	1	e
q_0	$\{q_0\}$	$\{q_0, q_2\}$	$\{q_1\}$
q_1	$\{q_5\}$	$\{q_2\}$	-
q_2	$\{q_3\}$	-	-
q_3	-	-	$\{q_4\}$
q_4	$\{q_3\}$	-	-
q_5	-	$\{q_4\}$	-

Спершу побудуємо граф заданого в умові НСА:



Побудуємо рівносильний ДСА за наступним алгоритмом:

Крок 1. Нехай $Q_e = cl_e(\{q_0\})$ – початковий стан шуканого ДСА. Покладемо $Q' = \{Q_e\}$ і $F' = \emptyset$. Якщо $Q_e \cap F \neq \emptyset$, то додаємо стан Q_e до множини F' кінцевих станів.

Крок 2. Повторюємо наступні пункти доти, доки значення $\delta'(Q_\omega, a)$ не буде визначене для всіх станів $Q_\omega \in Q'$ і всіх $a \in \{0,1\}$:

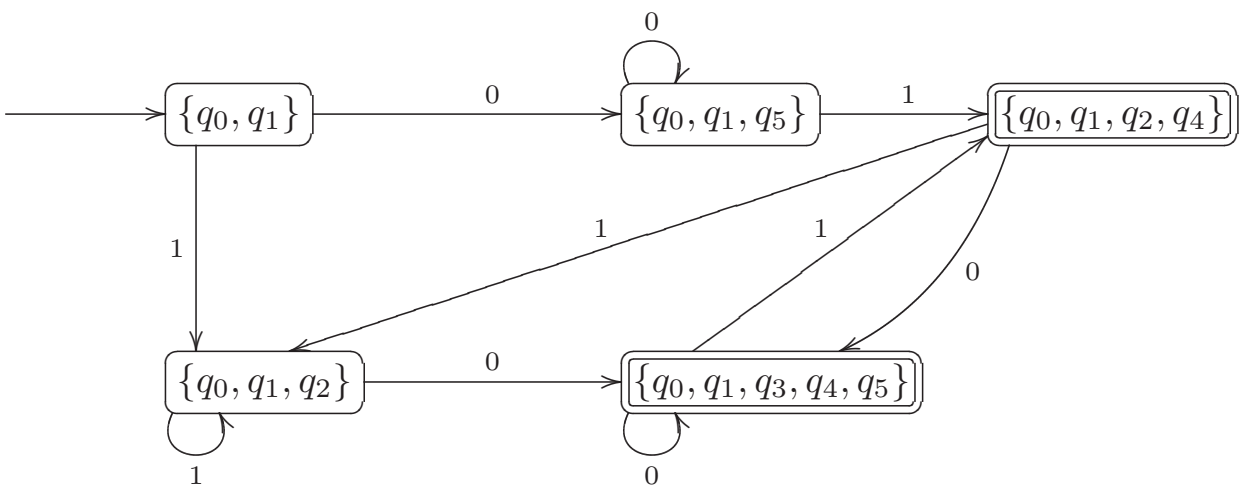
- Вибираємо такі $Q_\omega \in Q'$ і $a \in \{0,1\}$, що значення $\delta'(Q_\omega, a)$ функції переходів ще не визначене.
- Покладемо $Q_{\omega a} = \delta'(Q_\omega, a)$.
- Якщо $Q_{\omega a} \notin Q'$, то додаємо стан $Q_{\omega a}$ до множини станів Q', а також додаємо його до множини кінцевих станів F', якщо $Q_{\omega a} \cap F \neq \emptyset$.

Результат роботи алгоритму поданий у таблиці:

δ'	0	1
$Q_e = \{q_0, q_1\}$	$\{q_0, q_1, q_5\}$	$\{q_0, q_1, q_2\}$
$Q_0 = \{q_0, q_1, q_5\}$	$\{q_0, q_1, q_5\} = Q_0$	$\{q_0, q_1, q_2, q_4\}$
$Q_1 = \{q_0, q_1, q_2\}$	$\{q_0, q_1, q_3, q_4, q_5\}$	$\{q_0, q_1, q_2\} = Q_1$
$Q_{01} = \{q_0, q_1, q_2, q_4\}$	$\{q_0, q_1, q_3, q_4, q_5\}$	$\{q_0, q_1, q_2\} = Q_1$
$Q_{10} = \{q_0, q_1, q_3, q_4, q_5\}$	$\{q_0, q_1, q_3, q_4, q_5\} = Q_{10}$	$\{q_0, q_1, q_2, q_4\} = Q_{01}$

Зауважимо, що при виконанні кроку 2 не потрібно розглядати стани Q_{00}, Q_{000}, Q_{001} і т.д., бо $Q_{00} = Q_0$, $Q_{000} = Q_{00} = Q_0$ і $Q_{001} = Q_{01}$. Оскільки $Q_{11} = Q_1$, то не треба також розглядати стани $Q_{11\omega}$ для кожного $\omega \in \{0,1\}^*$.

Граф рівносильного до НСА ДСА має вигляд:



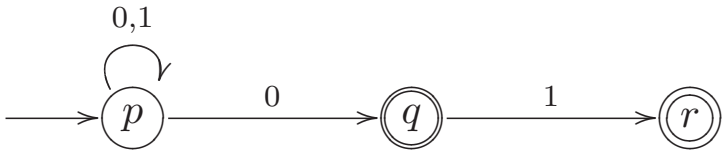
4.2. Приклад. Побудувати ДСА, який рівносильний до НСА

$$A = (\{p, q, r\}, \{0, 1\}, \delta, p, \{q, r\}),$$

де функція переходів δ задана таблицею:

δ	0	1
p	$\{p, q\}$	$\{p\}$
q	$-$	$\{r\}$
r	$-$	$-$

Зобразимо граф даного НСА:



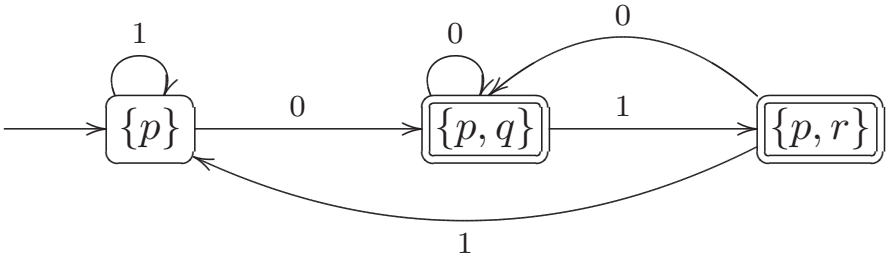
Використовуючи описаний в прикладі 4.1 алгоритм побудови рівносильного до НСА ДСА, одержимо наступну таблицю функції переходів ДСА:

δ'	0	1
$Q_e = \{p\}$	$\{p, q\}$	$\{p\} = Q_e$
$Q_0 = \{p, q\}$	$\{p, q\} = Q_0$	$\{p, r\}$
$Q_{01} = \{p, r\}$	$\{p, q\} = Q_0$	$\{p\} = Q_e$

Таким чином, шуканий ДСА задається як п'ятірка

$$A' = (\{\{p\}, \{p, q, \}, \{p, r\}\}, \{0, 1\}, \delta', \{p\}, \{\{p, q\}, \{q, r\}\}),$$

а його граф має вигляд:



З вищесказаного випливає, що для кожного НСА існує ДСА, який розпізнає ту ж мову, що і НСА. Оскільки кожен ДСА є частковим випадком НСА, то ми довели наступну теорему:

4.3. Теорема. *Формальна мова розпізнається НСА тоді і лише тоді, коли вона розпізнається ДСА.*

Теорема 4.3 дає простий метод побудови ДСА, який розпізнає мову L: спершу ми будуємо НСА для L, а потім перетворюємо його в рівносильний ДСА.

4.4. Приклад. Нехай $A = (\{p, q, r, s\}, \{0, 1\}, \delta, p, \{q, s\})$ – НСА, де

δ	0	1
p	$\{q, s\}$	$\{q\}$
q	$\{r\}$	$\{q, r\}$
r	$\{s\}$	$\{p\}$
s	$-$	$\{p\}$

Побудувати НСА, який розпізнає мову $\overline{L(A)}$.

Швидко побудувати шуканий НСА A' по заданому НСА A не вдасться, бо розглянутий в прикладі 2.10 для ДСА метод заміни некінцевих станів автомата A на кінцеві стани автомата A' в даному випадку незастосовний. Дійсно, можлива ситуація, коли для деякого слова ω обидва перетини $Q_\omega \cap F$ і $Q_\omega \cap (Q \setminus F)$ непорожні, і тому слово ω розпізнається обома автоматами A та A'. (Наприклад, при $Q_{01} = \{p, q, r\}$ і $F = \{q, s\}$ слово 01 розпізнається як A, так і A'.) Таким чином, $L(A') \neq \overline{L(A)}$.

Замість цього можна застосувати конструкцію підмножин для побудови шуканого НСА. Спершу знаходимо рівносильний до A ДСА A_D, а потім будуємо автомат A', граф якого такий же, як і автомата A_D, за винятком того, що кінцевими та некінцевими станами автомата A' є відповідно некінцеві та кінцеві стани ДСА A_D.

Використовуючи конструкцію підмножин побудуємо детермінований автомат $A_D = (Q_D, \{0, 1\}, \delta_D, \{p\}, F_D)$, де множина станів Q_D і функція переходів δ_D задані таблицею:

δ_D	0	1
$Q_e = \{p\}$	$\{q, s\}$	$\{q\}$
$Q_0 = \{q, s\}$	$\{r\}$	$\{p, q, r\}$
$Q_1 = \{q\}$	$\{r\}$	$\{q, r\}$
$Q_{00} = \{r\}$	$\{s\}$	$\{p\}$
$Q_{01} = \{p, q, r\}$	$\{q, r, s\}$	$\{p, q, r\}$
$Q_{11} = \{q, r\}$	$\{r, s\}$	$\{p, q, r\}$
$Q_{000} = \{s\}$	$\emptyset$	$\{p\}$
$Q_{010} = \{q, r, s\}$	$\{r, s\}$	$\{p, q, r\}$
$Q_{110} = \{r, s\}$	$\{s\}$	$\{p\}$
$Q_{0000} = \emptyset$	$\emptyset$	$\emptyset$

Множина F_D кінцевих станів має вигляд:

$$F_D = \{A \subset \{p, q, r, s\} \mid A \cap \{q, s\} \neq \emptyset\} =$$
$$\{\{q\}, \{s\}, \{q, s\}, \{q, r\}, \{r, s\}, \{p, q, r\}, \{q, r, s\}\}.$$

Таким чином, ДСА

$$A' = (Q_D, \{0, 1\}, \delta_D, \{p\}, Q_D \setminus F_D)$$

розпізнає мову $\overline{L(A)}$, де $Q_D \setminus F_D = \{\{p\}, \{r\}, \emptyset\}$.

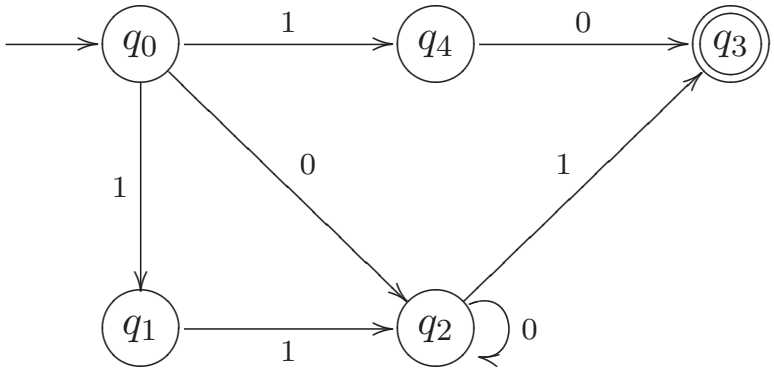
В параграфі 3 першого розділу для регулярного виразу було визначено його помічений граф $G(r)$ так, що слово $\omega \in L(r)$ тоді і лише тоді, коли існує шлях в $G(r)$ з початкової вершини в кінцеву вершину, позначки якого утворюють слово ω. Кожна стрілка графа $G(r)$ помічена однією буквою з множини $\{e\} \cup X$. Легко бачити, що $G(r)$ є графом НСА, який розпізнає мову $L(r)$. Таким чином, одержуємо твердження:

4.5. Твердження. *Кожна регулярна мова розпізнається деяким недетермінованим скінченним автоматом.*

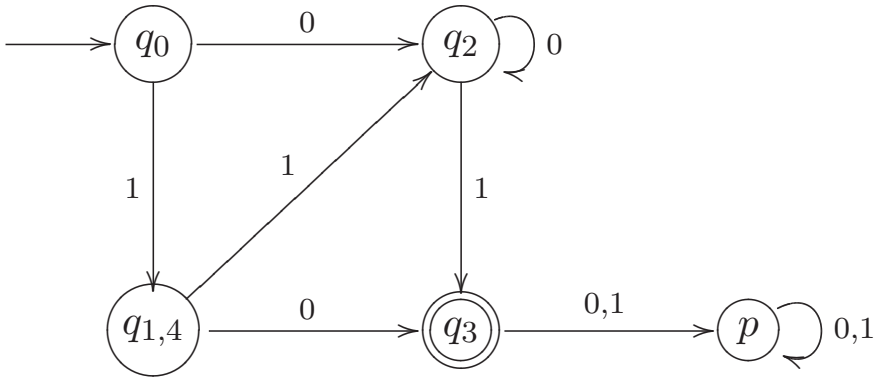
Розглянемо деякі приклади.

4.6. Приклад. Побудувати ДСА, який розпізнає регулярну мову, задану регулярним виразом $10 + (0 + 11)0^*1$.

Спершу побудуємо НСА, який розпізнає дану мову, використовуючи описаний в параграфі 3 першого розділу метод:



Далі перетворимо даний НСА в ДСА, використовуючи конструкцію підмножин. Граф шуканого ДСА має вигляд:

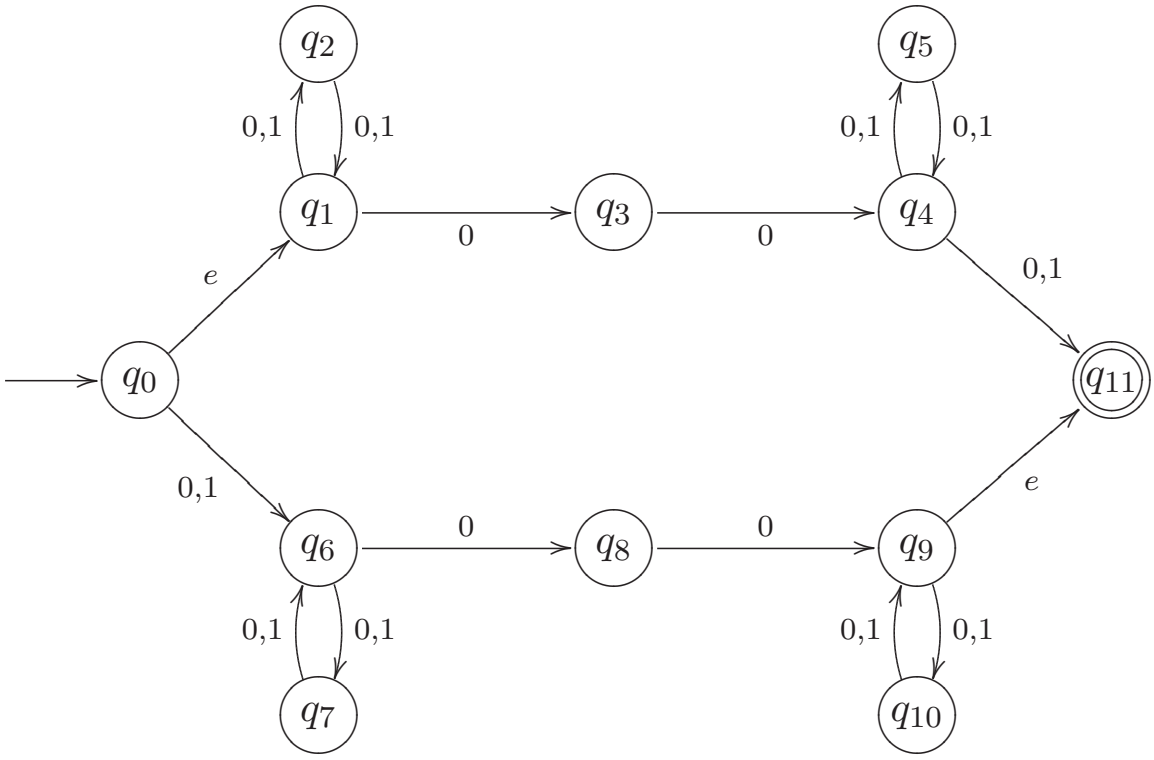


4.7. Приклад. Побудувати НСА, який розпізнає всі слова непарної довжини в бінарному алфавіті $X = \{0, 1\}$, які містять підслово 00.

Слово ω, яке містить підслово 00, може бути записане як $\omega = \alpha 00 \beta$ для деяких слів $\alpha, \beta \in \{0, 1\}^*$. Дане слово має непарну довжину тоді і лише тоді, коли довжини $|\alpha|$ і $|\beta|$ слів α і β мають різну парність. Таким чином, L запишеться наступним регулярним виразом:

$$((0+1)^2)^*00((0+1)^2)^*(0+1) + (0+1)((0+1)^2)^*00((0+1)^2)^*.$$

Використовуючи описаний в параграфі 3 першого розділу метод, побудуємо граф шуканого НСА:



Рекомендована література : [**2**, с. 138–140], [**4**, с. 521–531], [**12**, с. 429–434, 452–461], [**15**, с. 60–70].

Питання та вправи до параграфа 4.

4.1. Як побудувати ДСА, який рівносильний заданому НСА?

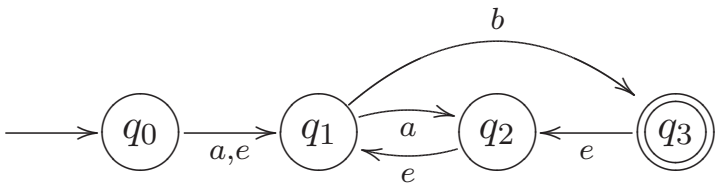
4.2. Як за заданим регулярним виразом r побудувати ДСА, який розпізнає формальну мову $L(r)$?

4.3. Побудувати ДСА, який розпізнає ту ж формальну мову, що і НСА $A = (Q, \{a, b\}, \delta, q_0, F)$, де $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $F = \{q_4\}$ і функція переходів δ задана таблицею:

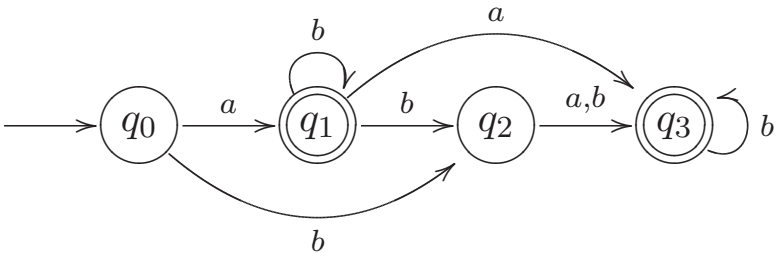
δ	a	b	e
q_0	$\{q_0, q_2\}$	$\{q_1\}$	$\{q_2\}$
q_1	$\{q_3\}$	$\{q_4\}$	-
q_2	-	$\{q_3, q_4\}$	-
q_3	$\{q_3\}$	-	$\{q_4\}$
q_4	$\{q_3\}$	$\{q_3\}$	-

4.4. Побудувати ДСА, які розпізнають ті ж формальні мови, що й НСА A та A', задані графами:

A :



A' :



4.5. Побудувати детерміновані скінченні автомати, які розпізнають мови $\overline{L(A)}$, $\overline{L(A')}$ та $\overline{L(A) \cap L(A')}$, де A та A' – автомати з попереднього прикладу.

4.6. Побудувати ДСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, що закінчуються на 00 або 11 або 10.

4.7. Побудувати ДСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, що починаються на 11 і закінчуються на 01.

4.8. Побудувати ДСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, в яких третьою буквою зліва та справа є 0.

4.9. Побудувати ДСА, який розпізнає всі слова в бінарному алфавіті $X = \{0, 1\}$, що містять одночасно 11 і 00, або не містять ні 00, ні 11.

4.10. Скласти регулярний вираз для множини всіх непарних натуральних чисел, записаних в четвірковій системі числення, та побудувати ДСА, який розпізнає дану множину.

4.11. Для кожного регулярного виразу r побудувати ДСА, який розпізнає регулярну мову $L(r)$:

а) $(0 + 10)^*(1 + 01)^*$;
б) $(0 + 1)^*0(0 + 1)(0 + 1)0(0 + 1)$;
в) $0(0 + 1)^*0 + 1(0 + 1)^*1$.

§ 5. Скінченні автомати і регулярні мови

В цьому параграфі ми покажемо, що скінченні автомати розпізнають виключно регулярні мови.

5.1. Теорема. *Для формальної мови L наступні умови рівносильні:*

1) *L – регулярна мова;*
2) *$L = L(A)$ для деякого детермінованого скінченного автомата A;*
3) *$L = L(A)$ для деякого недетермінованого скінченного автомата A;*
4) *мова L породжується деякою праволінійною граматикою.*

Доведення. Рівносильність 2) $\Leftrightarrow$ 3) доведена в теоремі 4.3, а імплікація 1) $\Rightarrow$ 3) в твердженні 4.5.

3) $\Rightarrow$ 1) Нехай формальна мова L розпізнається НСА $A = (Q, X, \delta, s, F)$. Покажемо, як побудувати регулярний вираз, що позначає мову $L(A) = L$. Спершу перетворимо автомат A до рівносильного автомата A' з єдиним кінцевим станом. Для цього замінимо кінцеві стани автомата A на некінцеві стани, додамо новий кінцевий стан $f \notin Q$ і проведемо e-стрілки з кожного старого кінцевого стану в новий кінцевий стан. Очевидно, що $L(A') = L(A)$. Граф автомата A' є поміченим графом G, кожна стрілка якого позначена єдиним символом з множини $\{e\} \cup X$. Такі графи розглядалися в параграфі 3 першого розділу.

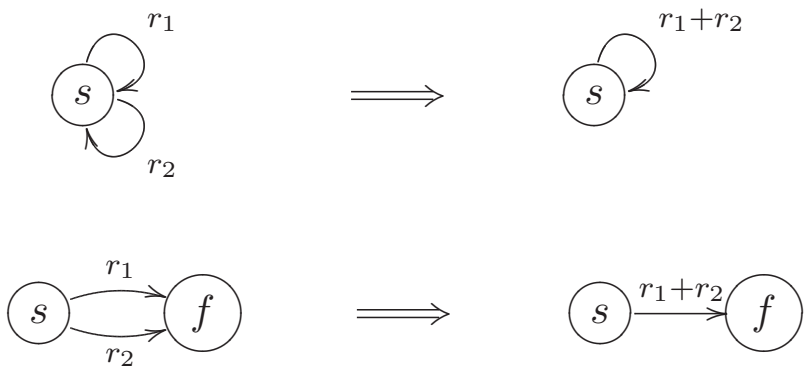
Для кожного шляху π з вершини s в вершину f через $r(\pi)$ позначимо регулярний вираз $r_1 r_2 \dots r_k$, утворений позначками стрілок з π:

$$s \xrightarrow{r_1} \nu_1 \xrightarrow{r_2} \nu_2 \xrightarrow{r_3} \cdots \xrightarrow{r_k} \nu_k = f$$

Очевидно, що $L(A) = \bigcup\{L(r(\pi)) \mid \pi -$ шлях з s до $f\}$.

Доведемо індукцією за кількістю вершин графа G, відмінних від s і f, що формальна мова $L(A)$ є регулярною мовою.

База індукції. Для $n = 0$ граф G має тільки дві вершини s та f або єдину вершину $s = f$. Замінимо всі паралельні стрілки з позначками $r_1, r_2, \dots, r_m$, які ідуть з вершини s в вершину f, на одну стрілку з позначкою $r_1 + r_2 + \dots + r_m$:

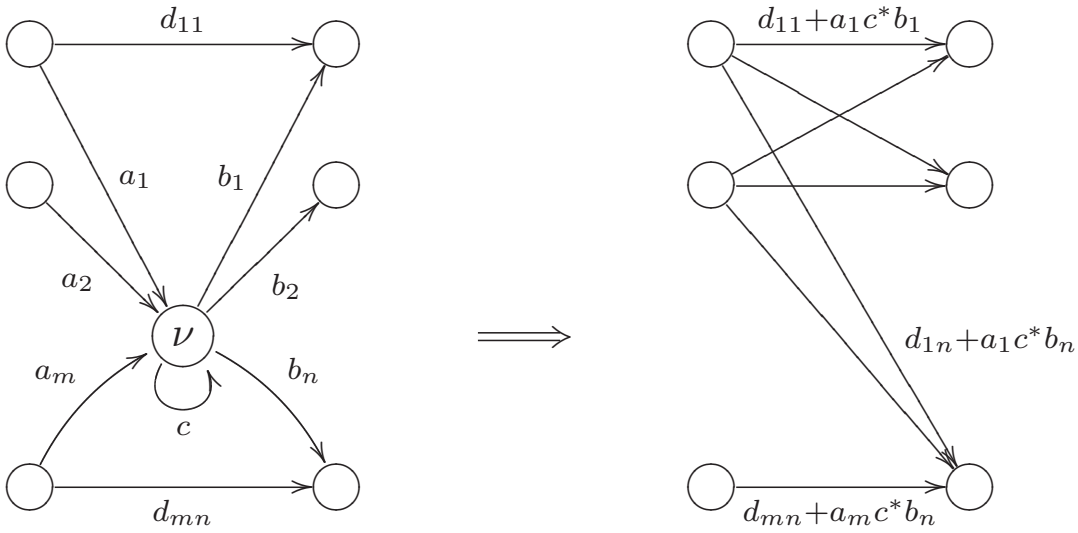


Як наслідок отримаємо один з двох графів:

де a, b, c, d – регулярні вирази.

Очевидно, що для першого графа G мова $L(A) = a^*$ і є регулярною. Розглянемо другий граф і шлях π з s до f. Припустимо, що шлях π проходить $k \geq 1$ разів через вершину f. Тоді π можна подати як послідовність $\pi_1\pi_2 \dots \pi_k$, де π_1 – це шлях з s в f, а $\pi_2, \dots, \pi_k$ – такі шляхи з f в f, що в кожному з них f не є проміжною вершиною. Очевидно, що $r(\pi_1) = a^{n_1}b$ для деякого $n_1 \geq 0$, а для $j = 2, \dots, k$ вираз $r(\pi_j)$ дорівнює c або $da^{n_j}b$ для деякого $n_j \geq 0$, тобто $r(\pi) \in a^*b(c + da^*b)^*$. Отже, $L(A) = a^*b(c + da^*b)^*$ і база індукції доведена.

Індуктивний крок. Для $n \geq 1$ оберемо відмінну від s та f вершину ν і видалимо її наступним чином. Спершу за використаним в базі індукції методом вилучимо паралельні стрілки. Таким чином, можна вважати, що існує не більше однієї стрілки з довільної вершини υ в довільну вершину ω. Далі припустимо, що ν має петлю з позначкою c. Видалимо вершину ν разом з цією петлею. Для довільної пари (υ, ω) вершин графа G з стрілками $\upsilon \xrightarrow{a} \nu$, $\nu \xrightarrow{b} \omega$, $\upsilon \xrightarrow{d} \omega$ видалимо стрілки $\upsilon \xrightarrow{a} \nu$ і $\nu \xrightarrow{b} \omega$ та замінимо позначку d стрілки $\upsilon \to \omega$ новою позначкою $d + ac^*b$ (якщо стрілки $\upsilon \to \omega$ в графі G не існує, то вважаємо, що вона має позначку $\emptyset$). Дане перетворення показано на рисунку:



(Щоб зробити даний рисунок простішим для розуміння, ми опустили деякі стрілки з розташованих ліворуч вершин у вершини, які розміщені праворуч. Вважаємо, що стрілка, яка веде з i-тої вершини зліва до j-тої вершини справа, помічена символом d_{ij}.) Зауважимо, що дане перетворення не змінює формальної мови $L(A)$. Таким чином, за індуктивним припущенням мова $L(A)$ є регулярною.

2) $\Rightarrow$ 4) Нехай мова L розпізнається ДСА $A = (Q, X, \delta, q_0, F)$. Без втрати загальності можна вважати, що $Q \cap X = \emptyset$. Побудуємо праволінійну граматику $G = (N, T, S, P)$, де $N = Q$, $T = X$, $S = q_0$,

$$P = \{q \to ap \mid \delta(q, a) = p\} \cup \{f \to e \mid f \in F\}.$$

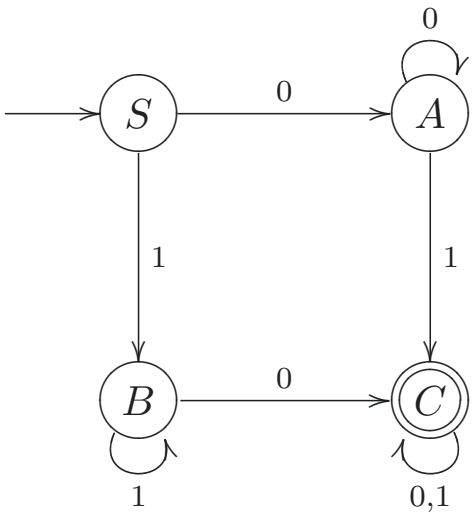
Математичною індукцією легко доводиться, що для будь-яких $p, q \in Q$ і кожного $\omega \in X$, $\delta(q, \omega) = p$ тоді і тільки тоді, коли існує виведення $q \Rightarrow^* \omega p$ в граматиці G. Зокрема, $\omega \in L$ тоді і лише тоді, коли існує виведення $q_0 \Rightarrow^* \omega f \Rightarrow \omega$ для деякого $f \in F$. Звідси випливає, що $L \subset L(G)$. Більше того, оскільки єдиними правилами в P, які в правій частині не містять нетермінальних символів, є $f \to e$, де $f \in F$, то вищезгадані виведення є єдиними в граматиці G. Таким чином, $L = L(G)$.

4) $\Rightarrow$ 3) Нехай $G = (N, T, S, P)$ – праволінійна граматика. Побудуємо помічений орієнтований граф наступним чином. Множина вершин співпадає з $N \cup \{f\}$, де $f \notin N$. Стан S є початковим станом, а f – єдиним кінцевим станом автомата. Для кожної продукції в G виду $A \to \omega B$, де $A, B \in N$ і $\omega \in T^*$ проводимо стрілку $A \xrightarrow{\omega} B$ в графі, а для кожної продукції вигляду $A \to \omega$, де $A \in N$ і $\omega \in T^*$ проводимо стрілку $A \xrightarrow{\omega} f$.

Легко бачити, що кожне виведення $S \Rightarrow^* \omega$ граматики G відповідає шляху графа від початкової вершини S до кінцевої вершини f, позначки якого утворюють слово ω. Таким чином, мова, яка відповідає даному поміченому графу, співпадає з мовою $L(G)$. Оскільки кожному такому поміченому орієнтованому графу відповідає НСА, який розпізнає ту ж мову, то мова $L(G)$ є регулярною.

□

5.2. Приклад. Побудуємо праволінійну граматику, яка породжується регулярною мовою, що розпізнається детермінованим скінченним автоматом, який заданий графом:



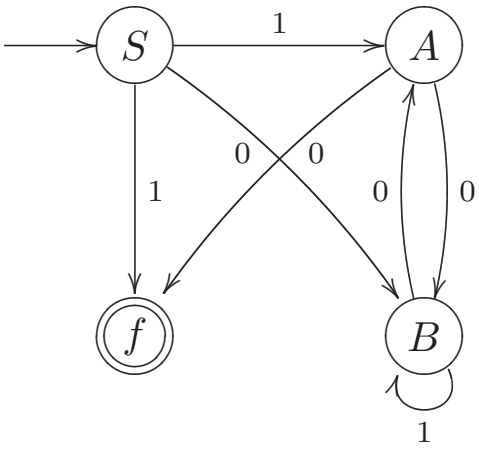
Використовуючи доведення теореми 5.1, побудуємо граматику $G = (N, T, S, P)$, де $N = \{S, A, B, C\}$, $T = \{0, 1\}$ і множина P продукцій містить наступні правила:

$$S \to 0A \mid 1B, \quad A \to 0A \mid 1C, \quad B \to 0C \mid 1B, \quad C \to 0C \mid 1C \mid e.$$

5.3. Приклад. Побудуємо недетермінований скінченний автомат, який розпізнає мову, що породжується праволінійною граматикою G, яка містить наступні продукції:

$$S \to 1A \mid 0B \mid 1, \quad B \to 0A \mid 1B, \quad A \to 0B \mid 0.$$

Граф автомата матиме вигляд:



Визначимо *частку формальних мов* L_1 *і* L_2 наступним чином:

$$L_1/L_2 = \{\omega \mid (\exists \nu \in L_2)[\omega\nu \in L_1]\}.$$

Наприклад, якщо

$L_1 = \{\omega \in \{0,1\}^* \mid \omega$ містить парну кількість входжень $0\}$,

$L_2 = \{0\}$ і $L_3 = \{0, 00\}$, то $L_1/L_3 = \{0,1\}^*$, а

$L_1/L_2 = \{\omega \in \{0,1\}^* \mid \omega$ містить непарну кількість входжень $0\}$.

5.4. Твердження. *Якщо* L_1 *і* L_2 *є регулярними мовами, то* L_1/L_2 *теж є регулярною мовою.*

Доведення. Нехай мова L_1 розпізнається детермінованим скінченним автоматом $A = (Q, X, \delta, q_0, F)$. Побудуємо автомат A', який розпізнає мову L_1/L_2. За означенням частки мов для слова $\omega \in L_1/L_2$ існує таке слово $\nu \in L_2$, що $\delta(q_0, \omega\nu) = \delta(\delta(q_0, \omega), \nu) \in F$. Нехай автомат A при аналізі слова ω переходить в стан $q = \delta(q_0, \omega)$. Якщо існує таке $\nu \in L_2$, що $\delta(q, \nu) \in F$, то слово ω має розпізнаватися A', а тому q слід визначити кінцевим станом автомата для мови L_1/L_2. Таким чином, покладемо

$$F' = \{q \in Q \mid (\exists \nu \in L_2)\ [\delta(q, \nu) \in F]\} \text{ і } A' = (Q, X, \delta, q_0, F').$$

Легко бачити, що A' розпізнає мову L_1/L_2.

□

З теорем 5.1, 2.11 і 3.7 та твердження 5.4 випливає наступний наслідок:

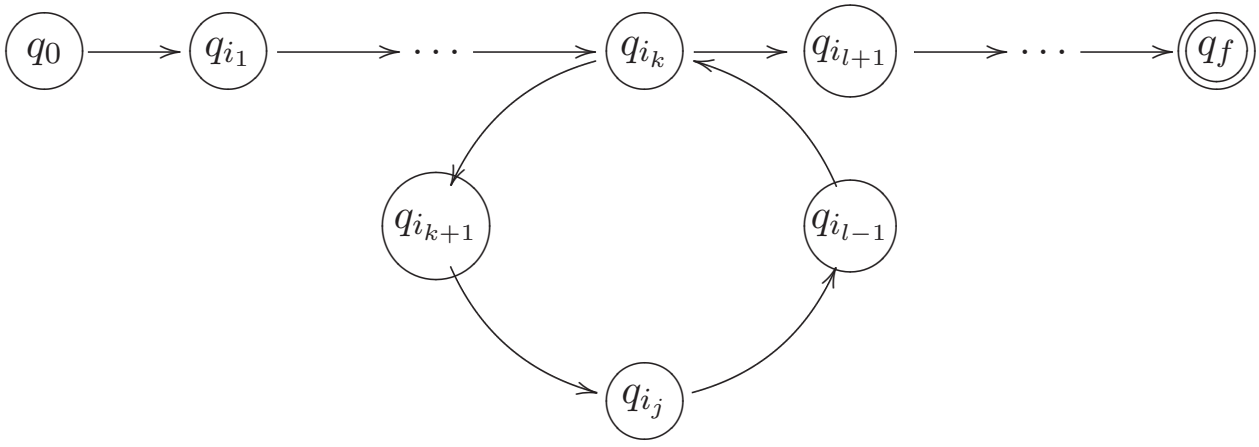
5.5. Наслідок. *Клас регулярних мов замкнений відносно скінченних об'єднань, перетинів, доповнення, різниці, симетричної різниці, конкатенації, ітерації та частки.*

З наслідку 5.5 випливає, що клас регулярних мов є досить широким. Природно виникає питання: чи даний клас співпадає з класом усіх формальних мов? Наступна лема стверджує, що це не так.

5.6. Лема. *(про роздування для регулярних мов) Нехай L – нескінченна регулярна мова, тоді існує така (залежна від L) константа s, що кожне слово $\omega \in L$, яке задовольняє нерівність $|\omega| \geq s$, можна так розбити на три підслова $\omega = \alpha\beta\gamma$, що виконуються наступні умови:*

а) $|\beta| \geq 1$;
б) $|\alpha\beta| \leq s$;
в) *для кожного $k \geq 0$ слово $\alpha\beta^k\gamma \in L$, тобто $\alpha\beta^*\gamma \subset L$.*

Доведення. Нехай L – регулярна мова, тоді згідно з теоремою 5.1 існує ДСА A, який розпізнає цю мову. Нехай кількість станів автомата A дорівнює s (константа, яка фігурує у формулюванні леми). Якщо $\omega \in L$, то обчислювальний шлях для слова ω починається в початковому стані q_0 і закінчується в кінцевому стані q_f. Цей шлях містить $|\omega|$ стрілок, бо кожна стрілка позначена єдиною буквою слова ω. Таким чином, він містить послідовність з $|\omega| + 1$ станів. Оскільки $|\omega| \geq s$, то згідно принципу Діріхле деякий стан в обчислювальному шляху обов'язково повториться двічі. Нехай $q_{i_k} = q_{i_l}$ – перший такий стан.



Нехай β – та частина слова ω, яка складається з букв, які позначають стрілки підшляху від першого входження стану q_{i_k} до другого його входження в обчислювальний шлях слова ω. Позначимо

через α частину слова ω перед β, а через γ – його частину після β. Очевидно, що $|\beta| \geq 1$ та $|\alpha\beta| \leq s$ (оскільки всі стани до другої появи q_{i_k} різні). Більше того, слово $\alpha\beta^k\gamma$ для будь-якого цілого невід'ємного числа k переводить автомат у той самий кінцевий стан, що й слово $\alpha\beta\gamma$. Справді, частина β^k слова $\alpha\beta^k\gamma$ просто переводить стани автомата A вздовж петлі, яка починається та закінчується в стані q_{i_k}. Ця петля проходить k разів. Звідси випливає, що автомат A розпізнає слова $\alpha\beta^k\gamma$, тому всі такі слова належать мові L.

□

5.7. Приклад. Доведемо, що мова $\{0^m1^m \mid m = 0, 1, \ldots\}$ не регулярна.

Припустимо, що мова L є регулярною. Тоді L розпізнається ДСА, який має s станів. Нехай слово $\omega = 0^m1^m$ і $|\omega| = 2m \geq s$. Тоді за лемою про роздування слова $\omega = 0^m1^m = \alpha\beta\gamma$ та $\alpha\beta^k\gamma$ належать мові L, причому $\beta \neq e$. Слово β не може містити водночас нулі й одиниці, бо тоді β^2 містило би 10 і $\alpha\beta^2\gamma$ не належало б мові L. Отже, слово β складається або виключно з нулів, або тільки з одиниць, але тоді $\alpha\beta^2\gamma$ містить або забагато нулів, або забагато одиниць і не належить мові L. Ця суперечність доводить, що мова $L = \{0^m1^m \mid m = 0, 1, \ldots\}$ не регулярна.

5.8. Приклад. Покажемо, що мова $\{0^p \mid$ p– просте число $\}$ не є регулярною.

Припустимо, що $L = \{0^p \mid$ p– просте число $\}$ є регулярною мовою. Тоді L розпізнається ДСА. Нехай s – кількість станів автомата. Зафіксуємо просте число $p > s$. Зауважимо, що $0^p \in L$ і $|0^p| = p > s$. Таким чином, за лемою про роздування 0^p можна записати як $0^p = \alpha\beta\gamma$, причому $|\beta| \geq 1$ і $\alpha\beta^*\gamma \subset L$.

Нехай $i = |\alpha| + |\gamma|$ та $j = |\beta|$. Тоді з умови $\alpha\beta^*\gamma \subset L$ випливає, що для кожного $k \geq 0$ слово $\alpha\beta^k\gamma = 0^{i+kj} \in L$, тобто для кожного $k \geq 0$ число $i + kj$ є простим. Зокрема, при $k = 0$ маємо, що i є простим числом. Отже, $i \geq 2$. Поклавши $k = i$, отримаємо, що $i(1+j)$ – просте число. Тим не менше, оскільки $\beta \neq e$, то з $j = |\beta| \geq 1$ випливає, що $i(1 + j)$ є складеним. Дана суперечність доводить нерегулярність мови $L = \{0^p \mid$ p– просте число $\}$.

5.9. Приклад. Доведемо, що мова $L = \{0^m \mid$ m– складене число $\}$ не регулярна.

Припустимо, що дана мова L є регулярною, тоді з наслідку 5.5 випливає, що мови $\overline{L}$ та $\overline{L} \setminus \{0\} = \{0^p \mid \text{p}-\text{ просте число }\}$ є регулярними, що суперечить прикладу 5.8.

Рекомендована література : [**2**, с. 141–147], [**4**, с. 509–521, 538–543], [**15**, с. 82–88, 114–123], [**16**, с. 294–303], [**22**, с. 53–69].

Питання та вправи до параграфа 5.

5.1. З яким класом формальних мов співпадають мови, що розпізнаються ДСА або НСА?

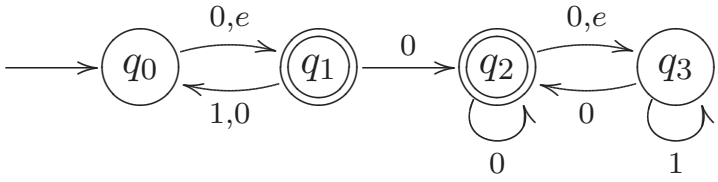
5.2. Дайте означення частки двох формальних мов.

5.3. Чи співпадає клас формальних мов з класом регулярних мов?

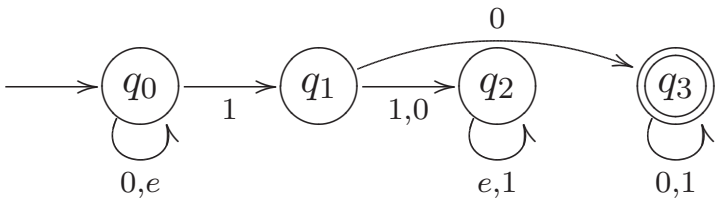
5.4. Сформулюйте лему про роздування для регулярних мов. Для чого її застосовують?

5.5. Побудувати праволінійну граматику, яка породжує мову, що розпізнається НСА без виходу:

а)



б)



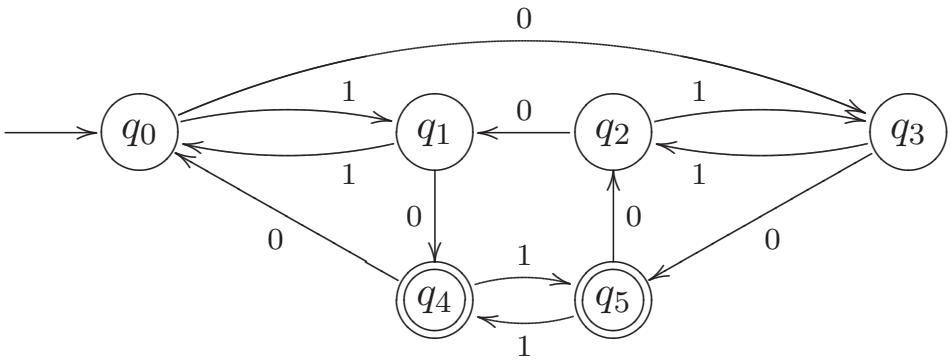
5.6. Побудувати граматику, яка породжує регулярну мову $L(r)$, задану регулярним виразом $r = ((0 + 11)^*10)^*$.

5.7. Побудувати НСА, який розпізнає мову, породжену граматикою G, що задана множиною продукцій P:

а) $P = \{S \to bB \mid aA, A \to a, B \to b\}$;
б) $P = \{S \to aB \mid aA, A \to aB \mid ab, B \to b\}$;
в) $P = \{S \to bA \mid B, A \to aaA \mid aa, B \to bB \mid b\}$.

5.8. Побудувати ДСА, який розпізнає мову, породжену граматикою G з попереднього прикладу.

5.9. Використовуючи доведення теореми 5.1, побудувати регулярний вираз, який позначає регулярну мову, що розпізнається автоматом:



5.10. Довести або спростувати твердження:

а) якщо мова A є регулярною і $A \subset B$, то мова B теж є регулярною;

б) якщо мова A є регулярною і $B \subset A$, то мова B – регулярна;

в) якщо мова A^2 є регулярною, то мова A є регулярною;

г) якщо мови A і AB є регулярними, то мова B також регулярна;

д) якщо мови A і B є регулярними, то мова $\cup_{i=0}^{\infty}(A^i \cap B^i)$ теж є регулярною.

5.11. З'ясувати, чи є регулярними наступні формальні мови:

а) $\{0^{2n}1^n \mid n \geq 0\}$;

б) $\{0^{2^n} \mid n \geq 0\}$;

в) $\{0^n1^n2^n \mid n \geq 0\}$;

г) $\{0^{2n} \mid n \geq 0\}$;

д) $\{0^{n^2} \mid n \geq 0\}$;

е) $\{0^m1^n \mid m \neq n\}$;

є) $\{0^{pq} \mid p, q -$ прості числа$\}$;

ж) $\{0^m1^n \mid m, n \geq 0,\ m = 2n + 1\}$;

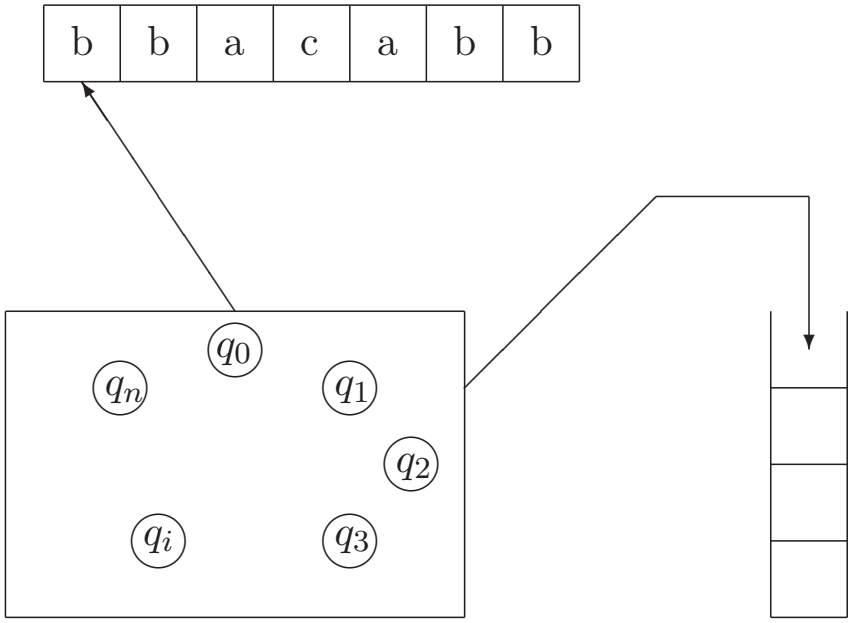
з) $\{0^m1^n \mid m, n \geq 0,\ m \neq 3n + 1\}$;

и) $\{a^nb^mc^k \mid n, m, k \geq 0,\ n \neq m$ або $m \neq k$ або $k \neq n\}$;

і) $\{\omega\omega^R \mid \omega \in \{0, 1\}^+\}$.

§ 6. Автомати з магазинною пам'яттю

Автомат з магазинною пам'яттю (МП-автомат) – це скінченний недетермінований автомат без виходу, що має додатковий пристрій пам'яті типу стек з головкою, яка виймає і заносить дані у стек. У стеку кожен елемент інформації заноситься в верхню комірку пам'яті, зміщуючи тим самим весь її вміст на одну комірку вниз, і дістається тільки із верхньої комірки пам'яті, зміщуючи весь її вміст на одну комірку вгору. При цьому у кожний момент часу доступний тільки верхній елемент стеку. Теоретично об'єм пам'яті МП-автомата необмежений. Схема МП-автомата має вигляд:



6.1. Означення. *Автомат з магазинною пам'яттю* – це шістка $M = (Q, X, Z, \delta, q_0, F)$, де Q, X, q_0 і F визначаються так само, як і в НСА, Z – скінченний магазинний алфавіт (множина допустимих букв пам'яті), а δ – функція переходів:

$$\delta : Q \times (X \cup \{e\}) \times (Z \cup \{e\}) \to 2^{Q \times (Z \cup \{e\})}.$$

На початку роботи МП-автомат $M = (Q, X, Z, \delta, q_0, F)$ завжди перебуває в початковому стані q_0, його стек порожній і головка стрічки аналізує першу зліва букву вхідного слова, яке записане на стрічці. Якщо МП-автомат M перебуває в стані q, зчитує з стрічки букву a і виймає зі стеку верхню букву u та $(p, v) \in \delta(q, a, u)$, то

M замінює u на v, головка стрічки переміщується на одну комірку праворуч і керуючий пристрій переводить автомат у стан p. Розглянемо випадки, коли v або u дорівнює e. Якщо $v = e$, то МП-автомат просто видаляє верхню букву u стеку, не виймаючи при цьому наступної букви. У випадку $u = e$ МП-автомат заносить в стек букву v, не зачіпаючи верхньої букви стеку. Команда $(p, v) \in \delta(q, e, u)$ аналогічна e-тактам НСА, тобто МП-автомат M виконує операцію, як і у випадку $(p, v) \in \delta(q, a, u)$, але при цьому не зчитує букви зі стрічки (головка стрічки не зміщується праворуч). Вважається, що МП-автомат M розпізнає вхідне слово, якщо хоча б в одному з обчислювальних шляхів він повністю проаналізовує слово, має порожній стек і зупиняється в кінцевому стані $q \in F$. Через $L(M)$ позначається мова, яка складається зі всіх слів, які розпізнаються автоматом M.

6.2. Приклад. Розглянемо МП-автомат

$$M = (\{q, p\}, \{a, b, c\}, \{a, b\}, \delta, q, \{p\}),$$

де функція переходів δ задається так:

$$\delta(q, a, e) = \{(q, a)\},\ \delta(p, a, a) = \{(p, e)\},\ \delta(q, b, e) = \{(q, b)\},$$
$$\delta(p, b, b) = \{(p, e)\}, \delta(q, c, e) = \{(p, e)\}.$$

Знайдемо мову $L(M)$.

Цей автомат працює наступним чином. В початковому стані q він заносить букви a та b в стек доти, доки головка стрічки не знайде букву c. Після аналізу букви c він переходить в стан p. В стані p автомат M порівнює кожну букву стрічки з верхньою буквою стеку. Якщо всі відповідні букви стрічки та стеку співпадають, то M розпізнає вхідне слово.

Нехай $\omega \in \{a, b\}^*$ – префікс вхідного слова перед першим входженням букви c і ν – суфікс вхідного слова після першого входження c. Зауважимо, що в той час, коли МП-автомат переходить в стан p, слово ω записане в стеку в оберненому порядку (перша буква слова ω записана вкінці стеку, а остання буква – на верху стеку). Таким чином, коли автомат порівнює суфікс ν з буквами стеку, то спершу порівнюється перша буква слова ν з останньою буквою слова ω і т.д. Отже, вхідне слово розпізнається тільки тоді, коли $\nu = \omega^R$, тобто $L(M) = \{\omega c \omega^R \mid \omega \in \{a, b\}^*\}$.

Щоб краще зрозуміти як автомат працює на вхідному слові, введемо поняття *конфігурації МП-автомата*. Конфігурація – це запис інформації про роботу автомата на певному кроці аналізу вхідного слова, який включає активний стан, букви стрічки, які ще не проаналізовані, і символи, які на даний момент є в стеку. Отже, конфігурація МП-автомата $M = (Q, X, Z, \delta, q_0, F)$ є елементом декартового добутку $Q \times X^* \times Z^*$. Конфігурація (q, ω, γ) означає, що МП-автомат перебуває в стані q, непроаналізований суфікс вхідного слова дорівнює ω (головка стрічки аналізує першу букву слова ω) і стек містить слово γ (перша буква слова γ є верхньою буквою стеку).

Нехай (q, ω_1, β) і (p, ω_2, γ) – дві конфігурації. Кажемо, що *конфігурація (p, ω_2, γ) безпосередньо слідує за (q, ω_1, β)* і записуємо $(q, \omega_1, \beta) \vdash (p, \omega_2, \gamma)$, якщо існує така інструкція $(p, v) \in \delta(q, a, u)$, що $\omega_1 = a\omega_2$ і $\beta = u\alpha$, $\gamma = v\alpha$ для деякого слова $\alpha \in Z^*$, де a, u, v можуть дорівнювати порожньому слову e. Будемо казати, що *конфігурація (p, ω_2, γ) слідує за (q, ω_1, β)* і записувати $(q, \omega_1, \beta) \vdash^* (p, \omega_2, \gamma)$, якщо існує така послідовність конфігурацій $C_0, C_1, \ldots, C_n$, що $C_0 = (q, \omega_1, \beta)$, $C_n = (p, \omega_2, \gamma)$ і $C_i \vdash C_{i+1}$ для всіх $i = 0, 1, \ldots, n-1$. Послідовність $(C_0, C_1, \ldots, C_n)$ конфігурацій МП-автомата M називається *обчислювальним шляхом автомата M*, якщо C_0 є початковою конфігурацією (q_0, ω, e), C_n не має наступної конфігурації і $C_i \vdash C_{i+1}$ для всіх $i = 0, 1, \ldots, n-1$. Нагадаємо, що в загальному випадку M є недетермінованим автоматом, а тому може бути багато обчислювальних шляхів, які починаються з початкової конфігурації. Крім того, конфігурація (q, ω, β) може не мати наступної конфігурації навіть при $\omega \neq e$. Таким чином, МП-автомат M розпізнає слово ω тоді і лише тоді, коли існує обчислювальний шлях автомата M, який починається з початкової конфігурації (q_0, ω, e) і закінчується конфігурацією (f, e, e) для деякого кінцевого стану $f \in F$, тобто

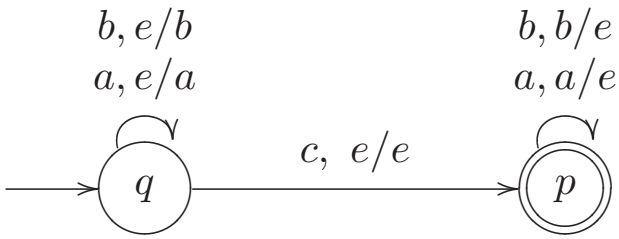
$$L(M) = \{\omega \in X^* \mid (\exists f \in F)[(q_0, \omega, e) \vdash^* (f, e, e)]\}.$$

Наприклад, розглянемо три обчислювальні шляхи МП-автомата M з прикладу 6.2 для слів $abcba$, $abcb$ і $abcbab$ відповідно:
$(q, abcba, e) \vdash (q, bcba, a) \vdash (q, cba, ba) \vdash (p, ba, ba) \vdash (p, a, a) \vdash (p, e, e)$;
$(q, abcb, e) \vdash (q, bcb, a) \vdash (q, cb, ba) \vdash (p, b, ba) \vdash (p, e, a)$;

$(q, abcbab, e) \vdash (q, bcbab, a) \vdash (q, cbab, ba) \vdash (p, bab, ba) \vdash (p, ab, a) \vdash (p, b, e)$.

Зауважимо, що з допомогою другого обчислювального шляху слово $abcb$ не розпізнається автоматом M, бо стек не стає порожнім після того, як слово проаналізується автоматом. Третій шлях теж не розпізнає вхідного слова, оскільки після (p, b, e) не існує наступної конфігурації, а вхідне слово ще не проаналізоване повністю.

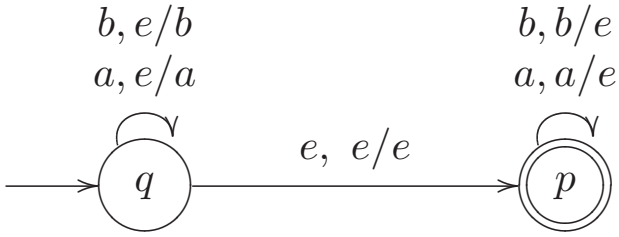
Аналогічно як і НСА, МП-автомат зображується графом. Вершини графа є станами автомата, а кожна стрілка з вершини q в вершину p з позначкою „$a, u/v$" зображує перехід $(p, v) \in \delta(q, a, u)$. Наприклад, граф автомата з прикладу 6.2 має вигляд:



6.3. Приклад. Побудувати МП-автомат, який розпізнає мову

$$\{\omega\omega^R \mid \omega \in \{a, b\}^*\}.$$

Цей приклад аналогічний до попереднього, за винятком того, що вхідне слово не містить всередині букви c і тому не зрозуміло, коли починати порівняння між буквами вхідного слова і буквами стеку. Як же вирішити цю проблему? Нагадаємо, що МП-автомат є НСА, а тому можна в будь-який час починати порівняння. Вхідне слово аналізується доти, доки вхідні букви, що залишилися, зіставляються коректно з буквами стеку по відношенню до деякої точки поділу. Звідси слідує, що шуканий МП-автомат задається графом:



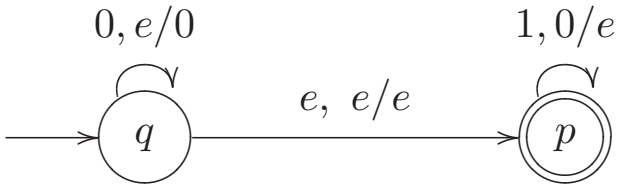
Побудуємо три обчислювальні шляхи для слова $abba$, перший з яких розпізнає дане слово, а два інші – ні:
$(q, abba, e) \vdash (q, bba, a) \vdash (q, ba, ba) \vdash (p, ba, ba) \vdash (p, a, a) \vdash (p, e, e)$;
$(q, abba, e) \vdash (q, bba, a) \vdash (q, ba, ba) \vdash (q, a, bba) \vdash (p, a, bba)$;

$(q, abba, e) \vdash (q, bba, a) \vdash (p, bba, a)$.

6.4. Приклад. Побудувати МП-автомат, який розпізнає мову

$$\{0^n 1^n \mid n = 0, 1, \ldots\}.$$

Користуючись тим же методом, що і в прикладі 6.3, побудуємо граф шуканого автомата:



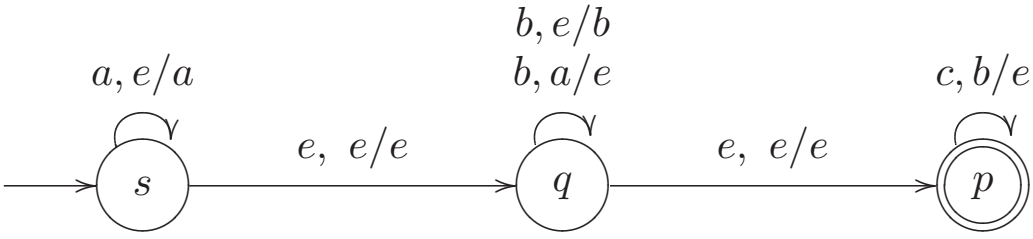
Даний автомат зчитує у пам'ять першу половину вхідного слова, що складається з нулів, а потім видаляє з пам'яті по одному нулю на кожну одиницю, що поступає на вхід. Крім того, переходи станів гарантують, що всі нулі передують всім одиницям.

З прикладів 5.7 і 6.4 випливає, що клас мов, які розпізнаються МП-автоматами, ширший від класу регулярних мов. Дані мови називаються *контекстно-вільними*.

6.5. Приклад. Побудувати МП-автомат, який розпізнає мову

$$\{a^i b^j c^k \mid i, j, k \geq 0, i + k = j\}.$$

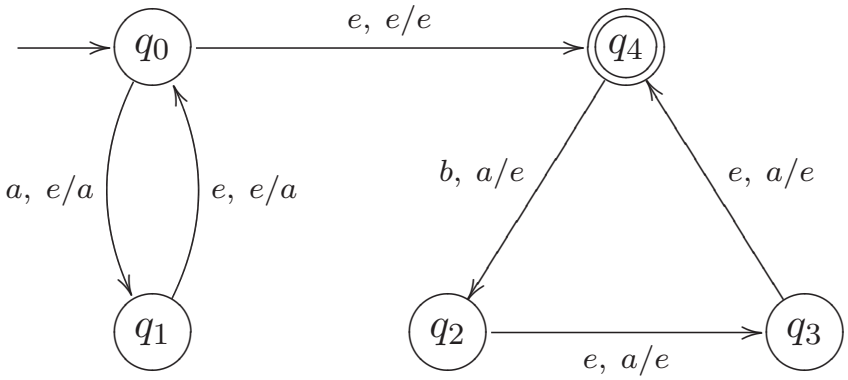
Основна ідея для розв'язання цієї задачі полягає у використанні різних станів для того, щоб зберігати правильний порядок входжень букв a, b і c, а також у використанні стеку для дотримання виконання рівності $i + k = j$. Таким чином, побудуємо МП-автомат з трьома станами s, q і p, де s – початковий стан, а p – єдиний кінцевий стан. В стані s автомат кожну букву a заносить у стек. В стані q він спершу взаємно однозначно співставляє букви b з буквами a зі стеку, а потім заносить решту букв b у стек. В стані p автомат аналізує букви c і співставляє їх з буквами b зі стеку.



6.6. Приклад. Побудувати МП-автомат, який розпізнає мову

$$L = \{a^i b^j \mid 2i = 3j\}.$$

Для того, щоб слово належало мові L, мають виконуватись рівності $i = 3k$ та $j = 2k$ для деякого цілого $k \geq 0$. Таким чином, для кожної букви a вхідного слова потрібно занести дві її копії в стек. Далі кожну букву b вхідного слова треба співставити з трьома буквами a стеку. Як же занести дві букви a у стек? Можна занести одну букву a у стек, а потім перейти в допоміжний стан для занесення другої букви a. Аналогічно спершу порівнюємо кожну букву b вхідного слова з однією буквою a, а потім переходимо в два інші допоміжні стани для видалення двох зайвих букв a зі стеку. Граф МП-автомата має вигляд:



Рекомендована література : [**2**, с. 192–220], [**5**, с. 399–402], [**11**, с. 374–382], [**15**, с. 128–136].

Питання та вправи до параграфа 6.

6.1. Дайте означення автомата з магазинною пам'яттю.

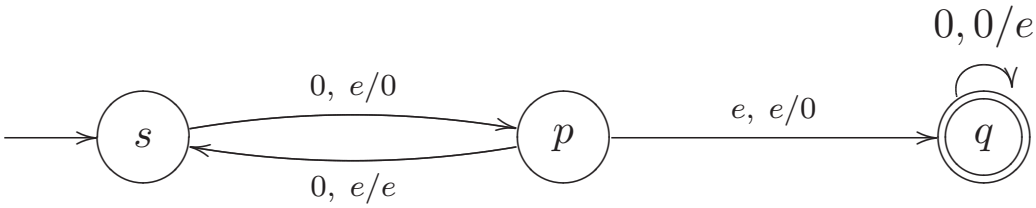
6.2. Чим відрізняється МП-автомат від НСА?

6.3. Що ви розумієте під стеком?

6.4. Обґрунтуйте, чому клас мов, що розпізнаються МП-автоматами, ширший від класу регулярних мов.

6.5. Як називаються мови, які розпізнаються МП-автоматами.

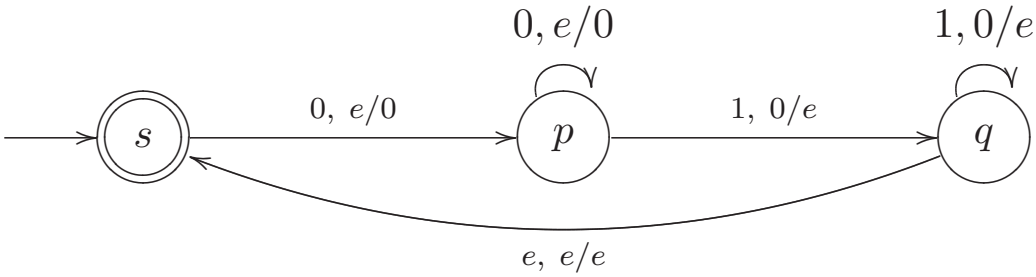
6.6. Скінченний МП-автомат M заданий графом:



Задати його як шістку $M = (Q, X, Z, \delta, q_0, F)$ і знайти мову, яка розпізнається МП-автоматом.

6.7. Чи є мова, що розпізнається МП-автоматом з попереднього прикладу, регулярною? Якщо так, то побудувати рівносильний ДСА.

6.8. Знайти мову $L(M)$, яка розпізнається МП-автоматом M, заданим графом:



Чи можна побудувати праволінійну граматику, яка породжує мову $L(M)$?

6.9. Побудувати МП-автомат, який розпізнає мову:

а) $\{0^n 1^m 2^n \mid n, m \geq 0\}$;
б) $\{a^i b^j c^k \mid i, j, k \geq 0, i + j = k\}$;
в) $\{a^i b^j c^k \mid i, j, k \geq 0, i = j + k\}$;
г) $\{a^i b^j c^k \mid i, j, k \geq 0, i + 2k = j\}$;
д) $\{0^n 1^{3n} \mid n, m \geq 0\}$;
е) $\{0^n 1^m \mid n, m \geq 0,\ 2n = 5m\}$.

Розділ III

Формальні алгоритмічні моделі

§ 1. Машина Тюрінґа

В попередньому розділі ми вивчили два типи обчислювальних моделей: скінченні автомати і автомати з магазинною пам'яттю, які розпізнають регулярні та контекстно-вільні мови відповідно. Зрозуміло, що дані моделі мають обмежені обчислювальні можливості в порівнянні з реальними персональними комп'ютерами. Наприклад, формальна мова $\{a^n b^n c^n \mid n \geq 0\}$ не розпізнається жодним із згаданих вище скінченних автоматів, але може бути легко обчислена з допомогою короткої програми на ПК.

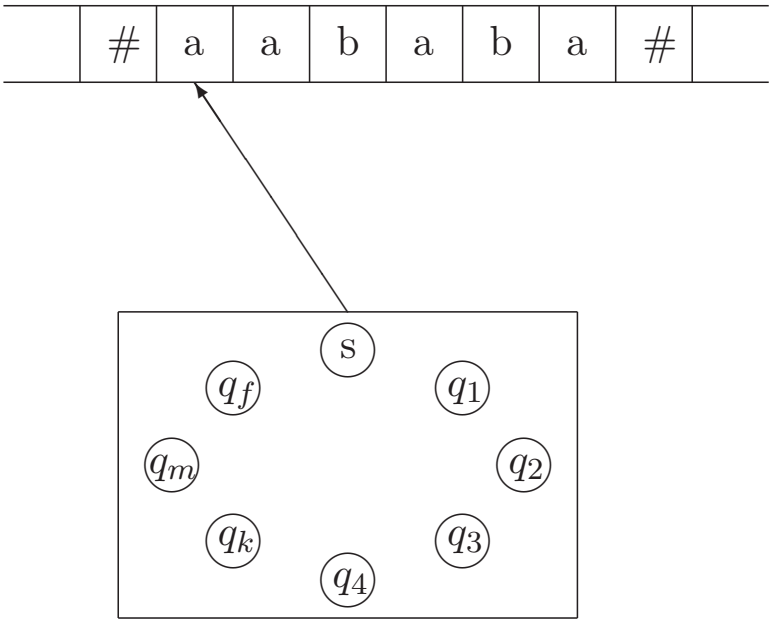
В цьому розділі ми опишемо нову обчислювальну алгоритмічну модель, яка називається машиною Тюрінґа (МТ). Основні відмінності між машиною Тюрінґа та скінченним автоматом полягають у тому, що стрічка МТ є нескінченною і МТ може пересуватись по стрічці (чи зміщувати стрічку) в будь-якому напрямку. Це надає машині потенційно необмежену пам'ять, яку можна використовувати під час обчислень.

МТ складається з керуючого пристрою, що містить головку і зовнішньої пам'яті (стрічки). Головка машини Тюрінґа може рухатися вздовж стрічки праворуч або ліворуч, читати та записувати символи, і, отже, поводиться аналогічно, як і справжній комп'ютер. Машина Тюрінґа має ті ж самі обчислювальні можливості, що і багато інших комп'ютерних моделей.

Розглянемо детальніше основні складові машини Тюрінґа.

Стрічка використовується для збереження інформації. Вона поділена на комірки і є нескінченною в обидва боки. Комірки не нумеруються. Кожна комірка містить єдиний символ із скінченного алфавіту A. Букви алфавіту $A = X \cup Y \cup D$ діляться на три групи. Букви алфавіту X називаються *вхідними буквами або символами* (з них утворюються слова, які записані на стрічці перед початком

роботи МТ). Через Y позначається множина букв, які називаються *вихідними буквами або символами* (з них утворюються слова, які записані на стрічці після завершення роботи МТ). D – множина *допоміжних символів*, до яких відноситься порожній символ #. Множини X та Y можуть перетинатися.



Головка машини Тюрінґа може рухатися вздовж стрічки комірка за коміркою праворуч або ліворуч. Крім того, вона може читати букву з комірки, яку аналізує, а також видаляти букву і записувати в комірку нову букву. В кожен момент часу головка розміщується під однією з комірок стрічки і аналізує її вміст. Дана клітинка називається *активною*, а символ, який в ній знаходиться – *активним символом*. Вміст клітинок, які не є активними, головка МТ не аналізує і не знає, які символи в них розміщені.

Головка машини Тюрінґа контролюється керуючим пристроєм, який складається зі скінченної кількості станів. Серед них є два виділені стани: початковий і кінцевий. Початковий стан зазвичай позначається через s або q_0, а кінцевий – через q_f. Множину станів позначимо через Q. В кожен момент часу керуючий пристрій перебуває в одному зі станів, які позначатимемо через $q_1, q_2, \ldots, q_n$. Перебуваючи в певному стані, МТ виконує деяку дію (наприклад,

рухається праворуч, міняючи при цьому вміст клітинок на порожній символ #). Після того, як головка прочитає символ зі стрічки, керуючий пристрій визначає, яку дію виконувати (який символ записувати і куди рухатися: праворуч чи ліворуч) і переходить в інший стан. МТ може виконувати тільки три елементарні дії: записувати в активну комірку новий символ, пересуватися на одну клітинку ліворуч або праворуч, переходити в новий стан. Жодних інших дій МТ виконувати не може! Тому більш складні дії повинні виражатися через елементарні. Ці дії описуються *функцією переходів (програмою)*:

$$\delta : Q \times A \to Q \times A \times \{L, R\}.$$

Якщо для функції δ виконується $\delta(q, a) = (p, b, R)$ ($\delta(q, a) = (p, b, L)$), то це означає, що перебуваючи в стані q, головка аналізує букву a (тобто буква a є активною), міняє її на b і рухається праворуч (відповідно ліворуч), керуючий пристрій при цьому переходить зі стану q у стан p. Описаний процес називається *тактом роботи машини Тюрінґа*. Таким чином, МТ працює тактами, що визначаються функцією δ.

Програма МТ записується у вигляді наступної таблиці:

δ	a_1	a_2	$\dots$	a_i	$\dots$	a_n	#
s							
q_1							
$\dots$							
q_j				$q', a', [L, R]$			
$\dots$							
q_m							

Зліва записуються всі стани, крім кінцевого q_f, з множини станів

$$Q = \{s, q_1, \dots, q_j, \dots, q_m, q_f\},$$

в яких може знаходитися керуючий пристрій МТ. Зверху – всі букви з алфавіту A (серед яких обов'язково є #), які головка МТ може аналізувати на стрічці. На перетинах рядків і стовпців таблиці вказуються ті такти, котрі повинен виконувати керуючий пристрій, знаходячись у відповідному стані й аналізуючи відповідну букву на стрічці.

Підсумовуючи все вищенаписане, машина Тюрінґа може бути визначена як п'ятірка $MT = (Q, A, \delta, s, q_f)$, де Q – скінченна множина станів, A – скінченний алфавіт, δ – функція переходів, $s \in Q$ – початковий стан, $q_f \in Q$ – кінцевий стан.

Вхідне слово – це скінченна послідовність букв з алфавіту $X \subset A$, записаних зліва направо по одній в кожній клітинці стрічки. Всередині вхідного слова порожніх клітинок немає. Всі клітинки ліворуч і праворуч від клітинок, заповнених буквами вхідного слова ω, вважаються порожніми, і в них знаходиться символ #. Якщо вхідне слово порожнє, то на стрічці всі клітинки заповнені порожнім символом #.

Для того щоб пояснити, як машина Тюрінґа працює з вхідним словом ω, визначимо поняття конфігурації МТ. Припустимо, що на деякому етапі роботи МТ перебуває в стані q, її стрічка містить символи $\ldots\#\#\#x_1x_2\ldots x_m\#\#\#\ldots$, де $x_i \in A$, $i = 1, \ldots, m$, і головка аналізує букву x_k. В випадку, коли $k \leq m$, будемо писати:

$$(q, x_1x_2\ldots x_{k-1}\underline{x_k}x_{k+1}\ldots x_m),$$

щоб позначити цю конфігурацію. Якщо ж $k > m$, то позначимо цю конфігурацію наступним чином:

$$(q, x_1x_2\ldots x_m \underbrace{\#\#\ldots\underline{\#}}_{k-m}).$$

Таким чином, конфігурація – це пара $(q, \omega) \in Q \times A^*$. Конфігурація $(q, x_1x_2\ldots x_{k-1}\underline{x_k}x_{k+1}\ldots x_m)$ позначає той факт, що машина перебуває в стані q, її стрічка містить слово $x_1x_2\ldots x_{k-1}x_kx_{k+1}\ldots x_m$ і головка аналізує букву x_k.

Вживаючи ці позначення, результат кожного руху МТ можна описати наступним чином:

- якщо $\delta(q, x_k) = (p, a, R)$, то після одного такту конфігурація $(q, x_1x_2\ldots\underline{x_k}x_{k+1}\ldots x_m)$ змінюється на конфігурацію $(p, x_1x_2\ldots a\underline{x_{k+1}}\ldots x_m)$;
- якщо $\delta(q, x_m) = (p, a, R)$, то після одного такту конфігурація $(q, x_1x_2\ldots\underline{x_m})$ змінюється на $(p, x_1x_2\ldots a\underline{\#})$;
- якщо $\delta(q, x_k) = (p, a, L)$, то після одного такту конфігурація $(q, x_1x_2\ldots x_{k-1}\underline{x_k}\ldots x_m)$ змінюється на конфігурацію $(p, x_1x_2\ldots\underline{x_{k-1}}a\ldots x_m)$;

- якщо $\delta(q, x_1) = (p, a, L)$, то після одного такту конфігурація $(q, \underline{x_1}x_2 \dots x_m)$ змінюється на $(p, \underline{\#}ax_2 \dots x_m)$;
- в конфігурації $(q_f, x_1x_2 \dots \underline{x_k}x_{k+1} \dots x_m)$ МТ зупиняє свою роботу і на виході дає слово $x_1x_2 \dots x_m$.

Якщо конфігурація $\alpha_1 = (q, \omega_1\underline{a}\omega_2)$ змінюється на конфігурацію $\alpha_2 = (q, \upsilon_1\underline{b}\upsilon_2)$, то писатимемо $\alpha_1 \to \alpha_2$. Якщо ми матимемо послідовність конфігурацій $\alpha_1, \alpha_2, \dots \alpha_n$, таких, що $\alpha_i \to \alpha_{i+1}$, $i = 1, \dots n-1$, то писатимемо $\alpha_1 \Rightarrow \alpha_n$.

На початку роботи машина Тюрінґа перебуває в початковому стані s і головка аналізує першу зліва непорожню клітинку. Далі починається виконання програми роботи МТ. У таблиці обирається клітинка на перетині першого рядка (бо керуючий пристрій знаходиться в початковому стані s) та того стовпчика, який відповідає першій букві вхідного слова, і виконується такт, вказаний в цій клітинці. Таким чином, МТ перейде до нової конфігурації. Потім виконуються такі ж дії, але вже в новій конфігурації: в таблиці знаходимо клітинку, що відповідає стану керуючого пристрою і букві цієї конфігурації, і виконується такт з цієї клітинки і т.д і т.п. Перейшовши в кінцевий стан q_f, МТ зупиняє свою роботу, і на виході одержуємо нове слово, яке записане в комірках стрічки МТ після завершення її роботи. Дане слово називатимемо *вихідним словом*. Кажемо, що *слово $\omega = x_1x_2 \dots x_m$ розпізнається машиною Тюрінґа*, якщо МТ, аналізуючи ω, зупиняє свою роботу в стані q_f, тобто $(s, \underline{x_1}x_2 \dots x_m) \Rightarrow (q_f, y_1y_2 \dots \underline{y_k} \dots y_n)$. В протилежному випадку слово ω не розпізнається МТ.

В момент зупинки всередині вихідного слова не може міститися порожніх клітинок, хоча в процесі виконання програми такі клітинки всередині проміжного слова можуть з'являтися.

Може трапитися так, що в процесі обробки вхідного слова ω, МТ ніколи не перейде в кінцевий стан q_f. У цьому випадку кажемо, що МТ *зациклюється на вхідному слові ω* і слово не розпізнається МТ. Якщо функція δ не визначена на наборі (q, a) і МТ, перейшовши в стан q, аналізує букву a слова ω, то у цьому випадку також будемо вважати, що слово ω не розпізнається МТ.

1.1. Приклад. Нехай машина Тюрінґа задана як п'ятірка $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_2, q_f\}$, $A = \{a, b, 0, 1, \#\}$, $X = \{a, b\}$, $Y = \{0, 1\}$, $D = \{\#\}$, і програма роботи δ задана у вигляді таблиці:

δ	a	b	$\#$
s	$s, 0, R$	$q_1, 1, R$	$q_2, \#, L$
q_1	$q_2, 0, R$	$q_1, 1, R$	$q_f, 0, L$
q_2	$q_2, 0, R$	$q_2, 1, R$	$q_2, 0, R$

Початкова конфігурація МТ на вхідному слові $aababa$ матиме вигляд $(s, \underline{a}ababa)$. Починаючи з цієї конфігурації, робота МТ на слові $aababa$ може бути описана наступним чином:

$(s, \underline{a}ababa) \to (s, 0\underline{a}baba) \to (s, 00\underline{b}aba) \to (q_1, 001\underline{a}ba) \to$
$(q_2, 0010\underline{b}a) \to (q_2, 00101\underline{a}) \to (q_2, 001010\underline{\#}) \to (q_2, 0010100\underline{\#}) \Rightarrow$
$(q_2, 001010\ldots0\underline{\#}) \to \ldots$

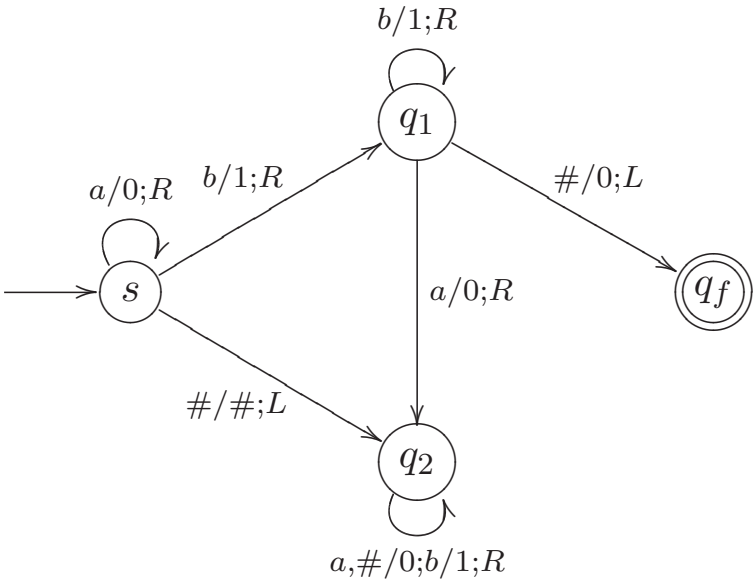
Таким чином, МТ ніколи не вийде зі стану q_2 і її головка нескінченно рухатиметься праворуч, міняючи при цьому порожні символи на 0. Іншими словами, в стані q_2 МТ зациклюється на вхідному слові $aababa$ і дане слово не розпізнається МТ.

На вхідному слові $aaaabb$ МТ працює іншим чином:

$(s, \underline{a}aaabb) \to (s, 0\underline{a}aabb) \to (s, 00\underline{a}abb) \to (s, 000\underline{a}bb) \to (s, 0000\underline{b}b)$
$\to (q_1, 00001\underline{b}) \to (q_1, 000011\underline{\#}) \to (q_f, 0000\underline{1}10)$.

Оскільки МТ перейшла в стан q_f, то вона зупиняє свою роботу. Вхідне слово $aaaabb$ розпізнається МТ. Вихідним словом є 0000110.

Для зручності МТ зображують поміченим орієнтованим графом $G(MT)$. Вершинами даного графа є стани автомата. Кожна стрілка графа має позначку виду „$a/b; R$" або „$a/b; L$". Якщо $\delta(q, a) = (p, b, R)$ $(\delta(q, a) = (p, b, L))$, то граф містить стрілку з позначкою „$a/b; R$" (відповідно „$a/b; L$"), початком якої є вершина q, а кінцем – вершина p. При цьому, якщо стрілка містить дві позначки виду „$a/b; R$" і „$c/b; R$" (відповідно „$a/b; L$" і „$c/b; L$"), то для компактності запису пишемо позначку „$a, c/b; R$" (відповідно „$a, c/b; L$"). Крім того, щоб підкреслити початковий стан s, малюється стрілка без початкової вершини, кінцевою вершиною якої є стан s. Кінцевий стан позначається двома концентричними колами. Наприклад, граф МТ з прикладу 1.1 має вигляд:



Мовою $L_1(MT)$, що розпізнається машиною Тюрінґа, називається множина всіх слів, які розпізнаються МТ, тобто

$$L_1(MT) = \{\omega \in A^* : \omega \text{ розпізнається МТ}\}.$$

Легко бачити, що в прикладі 1.1 мова, що розпізнається машиною Тюрінґа $L_1(MT) = \{a^m b^n : m \geq 0, n \geq 1\}$.

Кажемо, що формальна мова L розпізнається машиною Тюрінґа, якщо існує така МТ, що $L = L(MT)$.

Мовою $L_2(MT)$, що породжується машиною Тюрінґа, називається множина всіх вихідних слів, які утворюються з вхідних слів мови $L_1(MT)$, що розпізнається машиною Тюрінґа. Наприклад, в прикладі 1.1 мова, що породжується машиною Тюрінґа $L_2(MT) = \{0^m 1^n 0 : m \geq 0, n \geq 1\}$.

Рекомендована література : [**1**, с. 24–33], [**7**, с. 178–183], [**10**, с. 317–322], [**16**, с. 312–318], [**19**, с. 3–6], [**23**, с. 76–82].

Питання та вправи до параграфа 1.

1.1. Дайте означення машини Тюрінґа.

1.2. Що ви розумієте під тактом роботи МТ?

1.3. Опишіть процес обробки МТ вхідного слова.

1.4. Як будується граф МТ?

1.5. Нехай МТ задана як п'ятірка: $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_f\}$, $A = \{0, 1, \#\}$, а програма роботи δ задана у вигляді таблиці:

δ	0	1	#
s	$s, 1, R$	$q_f, 0, R$	$s, \#, R$

Виписуючи на кожному такті роботи МТ одержану конфігурацію, визначте, в яке слово перетворює МТ кожне з наступних вхідних слів:

а) 100;
б) 001;
в) 0010;
г) 0000.

1.6. Зобразіть граф МТ з попереднього прикладу та визначте мову, що розпізнається даною МТ.

1.7. В чому полягає робота машини Тюрінґа $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_f\}$, $A = \{0, 1, \#\}$, а програма роботи δ задана у вигляді таблиці:

δ	0	1	#
s	$s, 0, R$	$q_f, 1, R$	$q_f, 1, L$

?

1.8. Нехай МТ задана як п'ятірка: $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q, q_f\}$, $A = \{0, 1, \#\}$, а програма роботи δ задана у вигляді таблиці:

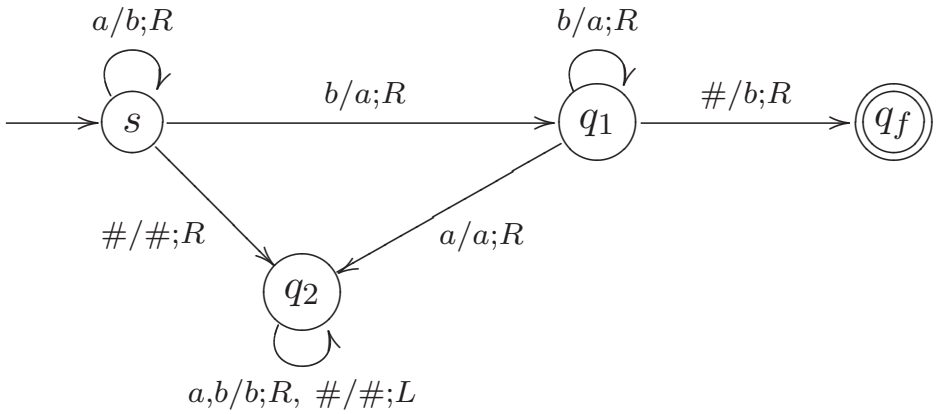
δ	0	1	#
s	$s, 0, R$	$s, 1, R$	$q, \#, L$
q	$q_f, 1, L$	$q, 0, L$	$q_f, 1, L$

Проаналізуйте роботу МТ на наступних вхідних словах:

а) 00;
б) 100;
в) 10011;
г) 111.

Вважаючи вхідні слова записами натуральних чисел в двійковій системі числення, вкажіть загальну закономірність роботи машини на усіх вхідних словах.

1.9. Нехай МТ зображена графом:



Задайте її як п'ятірку $MT = (Q, A, \delta, s, q_f)$ та визначте мову, що розпізнається даною МТ.

1.10. Знайти мову, що породжується машиною Тюрінґа з попереднього прикладу.

1.11. Побудувати граф та проаналізувати роботу машини Тюрінґа $MT = (Q, A, \delta, s, q_f)$ на словах $1^n = \underbrace{1 \ldots 1}_{n}$, де $Q = \{s, q_1, q_2, q_f\}$, $A = \{0, 1, \#\}$, а програма роботи δ задана у вигляді таблиці:

δ	0	1	#
s	$q_2, 1, R$	$q_1, \#, R$	$q_2, 1, R$
q_1	$q_2, 1, R$	$q_f, \#, R$	$q_2, \#, R$
q_2	$q_2, 1, R$	$q_2, 1, R$	$q_2, 1, R$

Як працює машина на слові 1000000? Яку мову розпізнає дана МТ?

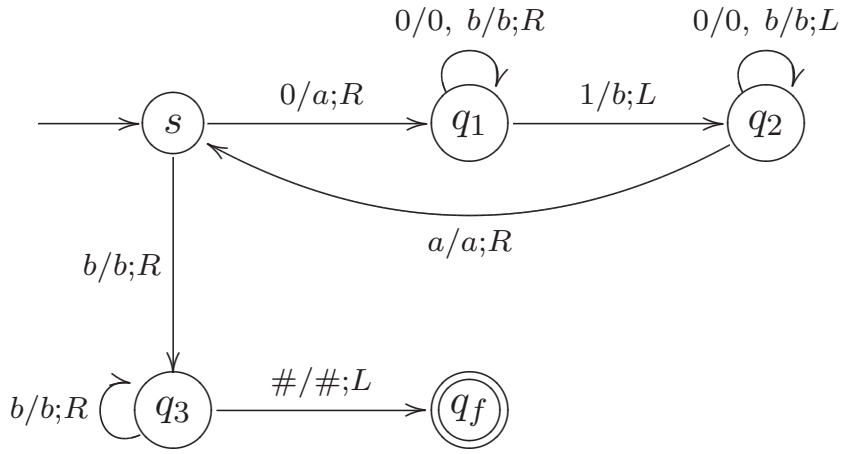
1.12. В чому полягає робота машини Тюрінґа $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_2, q_3, q_f\}$, $A = \{a, b, \#\}$, а програма роботи δ задана у вигляді таблиці:

δ	a	b	#
s	q_1, a, R	q_2, b, R	
q_1	q_1, a, R	q_2, a, R	$q_3, \#, L$
q_2	q_1, a, R	q_2, b, R	$q_3, \#, L$
q_3	q_f, a, R	$q_f, \#, L$	

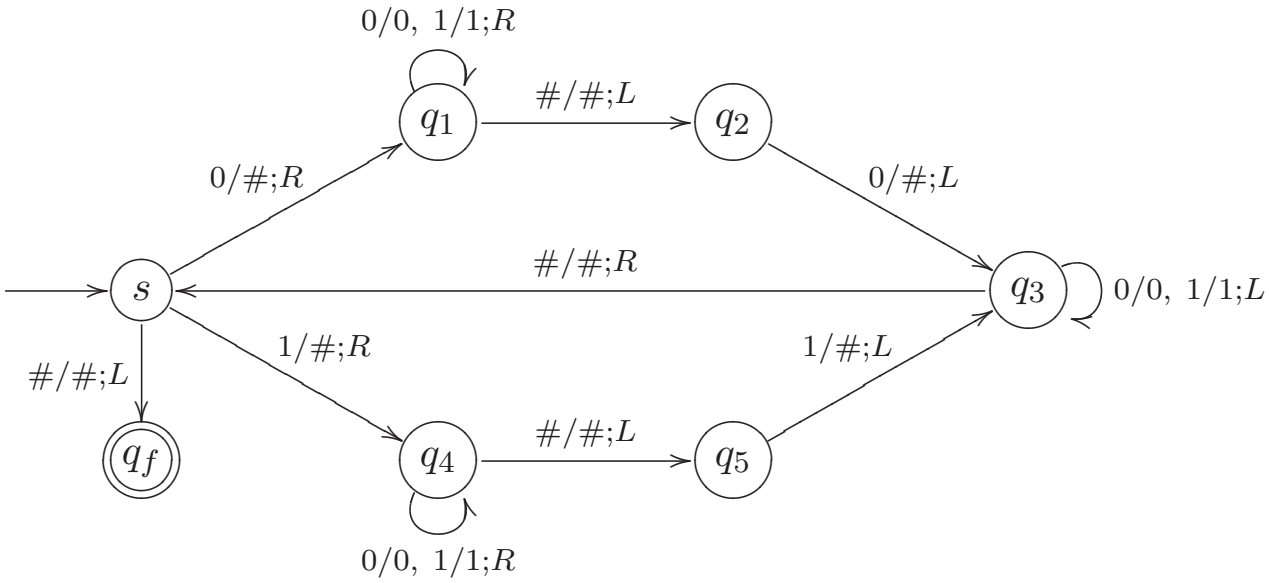
?

1.13. Знайти мову, яку розпізнає МТ, задана графом:

а)



б)



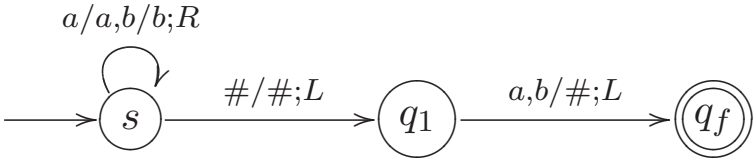
§ 2. Алгоритми синтезу МТ

У цьому параграфі на прикладах пояснюються основні прийоми складання алгоритмів для МТ.

2.1. Приклад. Побудувати машину Тюрінґа, яка розпізнає всі слова в алфавіті $X = \{a, b\}$ і видаляє із вхідного слова його останню букву.

Для розв'язання цієї задачі потрібно перемістити головку МТ під останню букву і видалити її. Нехай s – стан, в якому керуючий пристрій МТ переміщує головку комірка за коміркою праворуч, не змінюючи букви і залишаючись в стані s. Тут виникає невелика

проблема: МТ аналізує тільки активну комірку і не може визначити, чи наступна комірка є порожньою. Тому головка МТ спершу має перемістится під першу порожню комірку (в ній записаний символ #), яка розміщена після вхідного слова, а потім керуючий пристрій, перебуваючи в стані s і аналізуючи символ #, має перейти в новий стан q_1, в якому видалить останню букву і зупинить свою роботу (перейде в кінцевий стан q_f). Таким чином, граф буде мати наступний вигляд:



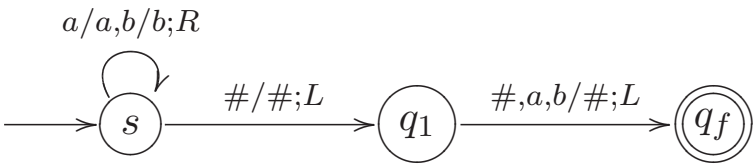
Проаналізуємо роботу даної МТ на вхідних словах abb, a і порожньому слові e.

$(s, \underline{a}bb) \to (s, a\underline{b}b) \to (s, abb) \to (s, abb\underline{\#}) \to (q_1, abb) \to (q_f, ab)$.

$(s, \underline{a}) \to (s, a\underline{\#}) \to (q_1, \underline{a}) \to (q_f, \underline{\#})$.

$(s, \underline{\#}) \to (q_1, \underline{\#})$

Ми бачимо, що на порожньому слові робота МТ невизначена і вона не розпізнає порожнього слова. В зв'язку з цим домовимося, що порожнє слово МТ не змінює. Остаточно, граф матиме вигляд:



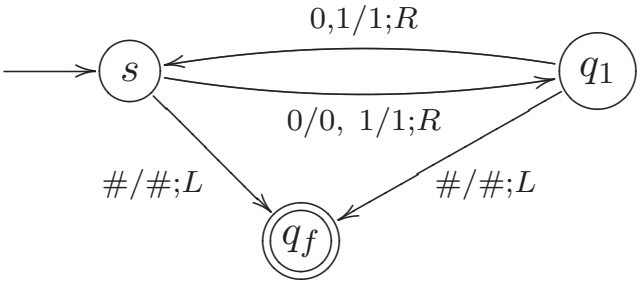
Таким чином, МТ задається як п'ятірка $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_f\}$, $A = \{a, b, \#\}$, $X = Y = \{a, b\}$, $D = \{\#\}$ і програма роботи δ задана у вигляді таблиці:

δ	a	b	$\#$
s	s, a, R	s, b, R	$q_1, \#, L$
q_1	$q_f, \#, L$	$q_f, \#, L$	$q_f, \#, L$

2.2. Приклад. Побудувати машину Тюрінґа, яка розпізнає всі слова в алфавіті $X = \{0, 1\}$ і замінює на 1 кожну другу букву вхідного слова.

Для розв'язання цієї задачі потрібно навчитися визначати, чи знаходиться головка МТ під буквою, яка міститься на парній позиції вхідного слова. Для цього розглянемо два стани s і q_1, в які

керуючий пристрій буде почергово переходити при русі головки МТ праворуч і аналізі слова буква за буквою, починаючи з першої букви. Таким чином, якщо МТ перебуває в стані s, то вона аналізує букву, що знаходиться на непарній позиції у вхідному слові; якщо ж МТ перебуває в стані q_1, то вона аналізує букву, що знаходиться на парній позиції. Домовимося також, що МТ не змінює порожнього слова. Отже, необхідно визначити роботу МТ так, щоб у стані s головка МТ не змінювала букви, а в стані q_1 замінювала 0 і 1 на 1. Підсумовуючи все вищесказане, побудуємо граф МТ наступним чином:



Шукана МТ задається як п'ятірка $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_f\}$, $A = \{0, 1, \#\}$, $X = Y = \{0, 1\}$, $D = \{\#\}$ і програма роботи δ задана у вигляді таблиці:

δ	0	1	#
s	$q_1, 0, R$	$q_1, 1, R$	$q_f, \#, L$
q_1	$s, 1, R$	$s, 1, R$	$q_f, \#, L$

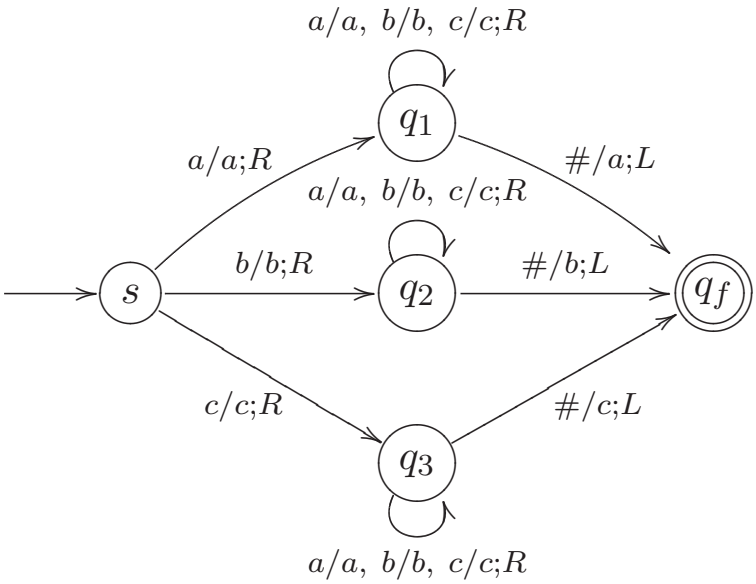
2.3. Приклад. Побудувати машину Тюрінґа, яка розпізнає всі непорожні слова в алфавіті $X = \{a, b, c\}$ і дописує вкінці вхідного слова його першу букву.

Для розв'язання даної задачі необхідно виконати наступні дії:

- запам'ятати першу букву;
- перемістити головку МТ праворуч під першу порожню комірку і записати в неї першу букву.

Як переміщуватися праворуч ми вже знаємо з попередніх прикладів. Але як запам'ятати першу букву? Адже в МТ немає іншої пам'яті, крім стрічки, а запам'ятовувати букву в якій-небудь комірці немає сенсу: як тільки головка МТ переміститься ліворуч чи праворуч від цієї клітки, МТ відразу ж забуде дану букву. Як же вийти з цієї ситуації? Стандартним вирішенням цієї проблеми є наступне – для кожної букви треба використовувати окрему множину

станів. Якщо першою буквою є a, то потрібно перейти в стан q_1, в якому головка МТ переміщується праворуч і записує вкінці слова букву a. Якщо ж першою буквою є буква b, то треба перейти в стан q_2, в якому виконуються ті ж самі дії, тільки вкінці слова дописується буква b. У випадку, якщо першою була буква c, переходимо в стан q_3, в якому дописуємо вкінці вхідного слова букву c. Цей прийом будемо використовувати і надалі для запам'ятовування різних букв вхідного слова. Підсумовуючи все сказане, граф МТ можна зобразити наступним чином:



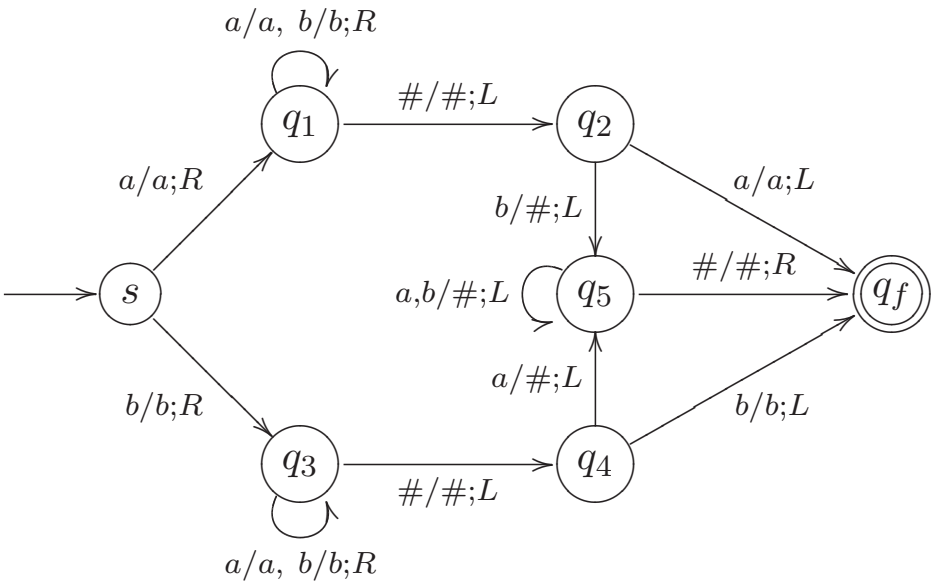
Програма роботи δ машини Тюрінґа $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_2, q_3, q_f\}$, $A = \{a, b, c, \#\}$, $X = Y = \{a, b, c\}$, $D = \{\#\}$, задається у вигляді таблиці:

δ	a	b	c	$\#$
s	q_1, a, R	q_2, b, R	q_3, c, R	$-$
q_1	q_1, a, R	q_1, b, R	q_1, c, R	q_f, a, L
q_2	q_2, a, R	q_2, b, R	q_2, c, R	q_f, b, L
q_3	q_3, a, R	q_3, b, R	q_3, c, R	q_f, c, L

2.4. Приклад. Побудувати машину Тюрінґа в алфавіті $X = \{a, b\}$, яка працює наступним чином: якщо перша і остання букви непорожнього вхідного слова співпадають, то слово залишається незмінним, в протилежному випадку замінює його на порожнє слово.

Для розв'язання цієї задачі потрібно запам'ятати першу букву вхідного слова, перемістити головку МТ під останню букву і порівняти її з першою. Якщо ці букви однакові, то МТ має перейти в

кінцевий стан і зупинити свою роботу, в протилежному випадку – видалити всі букви в слові. З попереднього прикладу ми вже знаємо, що для кожної букви потрібно створити свою множину станів для запам'ятовування даної букви і порівняння її з останньою буквою. Нехай такими станами для букви a будуть q_1, в якому МТ запам'ятовує a, і q_2, в якому перевіряється чи останньою є буква a. Аналогічні стани для букви b позначимо через q_3 і q_4. Якщо ж в стані q_2 головка МТ аналізує букву b або в стані q_4 головка МТ аналізує букву a, то ці букви видаляються (замінюються порожнім символом #) і МТ переходить в новий стан q_5, в якому головка МТ переміщується ліворуч на початок слова, видаляючи при цьому всі букви. Таким чином, граф МТ буде мати наступний вигляд:

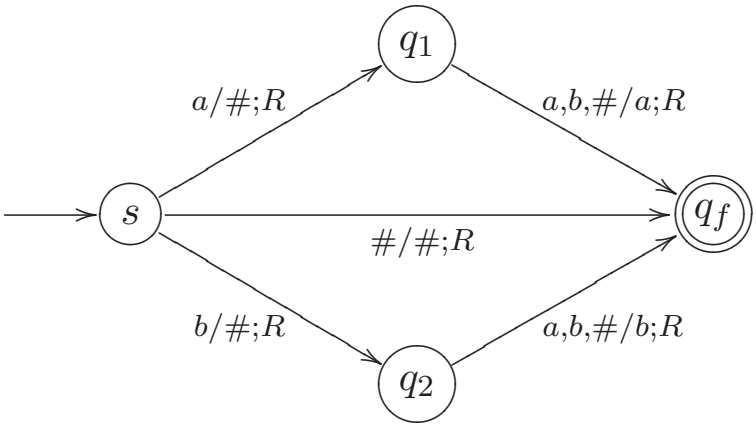


Отже, МТ задається як п'ятірка $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_f\}$, $A = \{a, b, \#\}$, $X = Y = \{a, b\}$, $D = \{\#\}$ і програма роботи δ задана у вигляді таблиці:

δ	a	b	$\#$
s	q_1, a, R	q_3, b, R	–
q_1	q_1, a, R	q_1, b, R	$q_2, \#, L$
q_2	q_f, a, L	$q_5, \#, L$	–
q_3	q_3, a, R	q_3, b, R	$q_4, \#, L$
q_4	$q_5, \#, L$	q_f, b, L	–
q_5	$q_5, \#, L$	$q_5, \#, L$	$q_f, \#, R$

2.5. Приклад. Побудувати машину Тюрінґа, яка розпізнає всі слова в алфавіті $X = \{a, b\}$ і видаляє із вхідного слова його другу букву.

На перший погляд здається, що таку МТ побудувати надзвичайно просто: достатньо змістити головку МТ на одну букву праворуч, а потім видалити другу букву вхідного слова. Але вкінці роботи МТ вихідне слово не може містити пропусків, тому після видалення другого символу потрібно стиснути слово, тобто перемістити першу букву в комірку праворуч. Проте, переміщуючись праворуч до другої букви, а потім повертаючись назад до першої букви, ми виконуємо зайву роботу: яка різниця чи переносити першу букву в порожню комірку чи в комірку з якоюсь буквою? Тому запам'ятовуємо першу букву, видаляємо і записуємо її замість другої букви. Крім того, вважатимемо, що МТ не змінює порожнього слова. Отже, граф МТ матиме наступний вигляд:



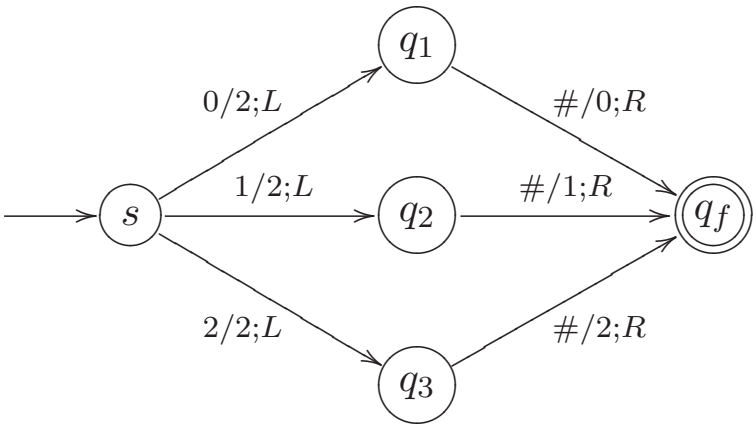
Проаналізуємо роботу МТ на вхідному слові, яке містить тільки одну букву. Аналізуючи дану букву, головка МТ видалить її, переміститься на одну комірку праворуч і запише в порожню комірку дану букву. Таким чином, слово із однієї букви просто зміститься на комірку праворуч. Це допустимо, бо комірки стрічки не нумеруються, тому розташування слова на стрічці ніяк не фіксується і переміщення слова ліворуч чи праворуч неможливо помітити. В зв'язку з цим не вимагається, щоб вихідне слово обов'язково знаходилось на тому ж місці, де було вхідне слово: результат може зміститися ліворуч чи праворуч від початкового місця.

Шукана МТ задається як п'ятірка $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_2, q_f\}$, $A = \{a, b, \#\}$, $X = Y = \{a, b\}$, $D = \{\#\}$ і програма роботи δ задана у вигляді таблиці:

δ	a	b	$\#$
s	$q_1, \#, R$	$q_2, \#, R$	$q_f, \#, R$
q_1	q_f, a, R	q_f, a, R	q_f, a, R
q_2	q_f, b, R	q_f, b, R	q_f, b, R

2.6. Приклад. Побудувати машину Тюрінґа, яка за першою буквою непорожнього слова в алфавіті $X = \{0, 1, 2\}$ вставляє букву 2.

Шукана МТ має спочатку вивільнити місце для букви 2. Для цього потрібно „розтиснути" слово. Ми можемо запам'ятати першу букву, видалити, записати її в комірці ліворуч, а потім у порожню комірку записати букву 2. Але, аналогічно як у попередньому прикладі, можна зробити це простіше: запам'ятати першу букву, записати на її місце букву 2, а потім переміститися ліворуч і записати в порожній комірці першу букву вхідного слова. Таким чином, отримуємо наступний граф МТ:



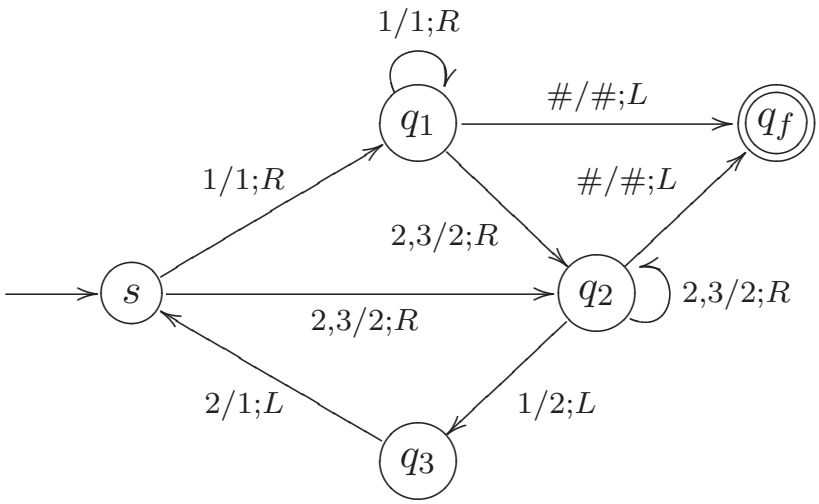
Отже, МТ задається як п'ятірка $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_2, q_3, q_f\}$, $A = \{0, 1, 2, \#\}$, $X = Y = \{0, 1, 2\}$, $D = \{\#\}$ і програма роботи δ задана у вигляді таблиці:

δ	0	1	2	$\#$
s	$q_1, 2, L$	$q_2, 2, L$	$q_3, 2, L$	–
q_1	–	–	–	$q_f, 0, R$
q_2	–	–	–	$q_f, 1, R$
q_3	–	–	–	$q_f, 2, R$

2.7. Приклад. Побудувати машину Тюрінґа в алфавіті $X = \{1, 2, 3\}$, яка у непорожньому вхідному слові замінює всі букви 3 на букви

2 і впорядковує послідовність букв 1 і 2 так, щоб спочатку були записані букви 1, а потім букви 2.

Шукана МТ має впорядкувати вхідне слово, замінивши при цьому всі букви 3 на букви 2. Принцип її роботи може бути описаний наступним чином. Як завжди, на початку роботи МТ знаходиться в початковому стані s і її головка аналізує першу букву вхідного слова. В стані q_1 головка МТ аналізує всі букви 1 слова, рухаючись праворуч доти, доки не знайде першу букву 2 або 3. Далі керуючий пристрій переходить в стан q_2, в якому головка МТ, рухаючись праворуч, аналізує букви 2 і 3, міняючи при цьому 3 на 2 доти, доки не побачить наступну 1. Після цього вона міняє місцями в стані q_3 1 з 2, що знаходиться ліворуч, і знову повертається в початковий стан, повторюючи далі аналогічні кроки аж доки вхідне слово не буде впорядковане. Граф МТ має вигляд:



Проаналізуємо роботу даної МТ на вхідному слові 132113.

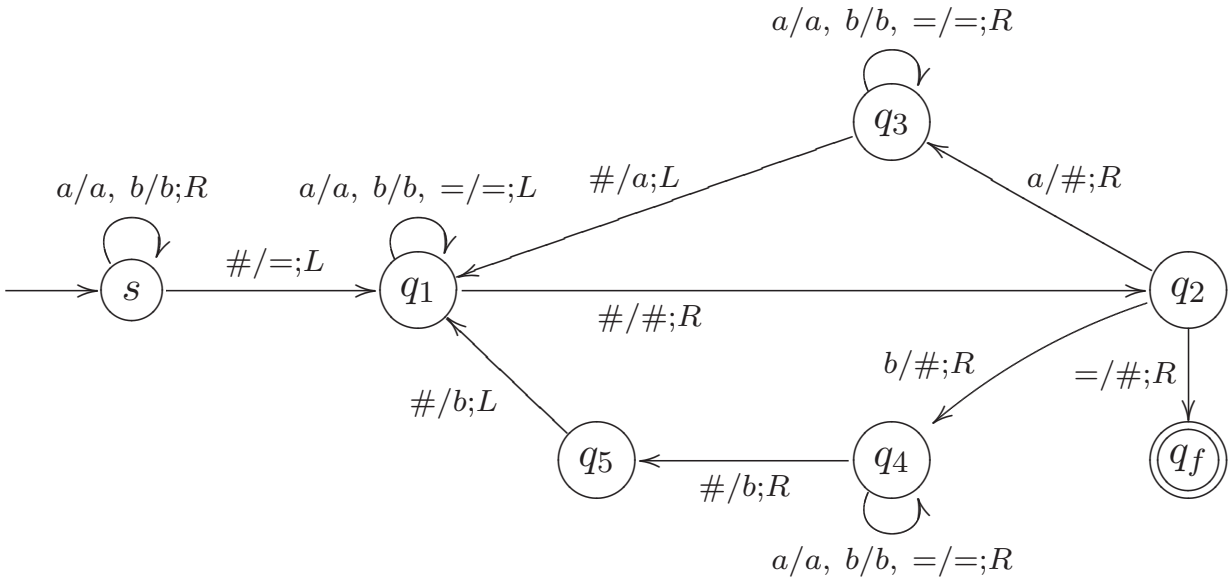
$(s, \underline{1}32113) \to (q_1, 1\underline{3}2113) \to (q_2, 12\underline{2}113) \to (q_2, 122\underline{1}13) \to$
$\to (q_3, 12\underline{2}213) \to (s, 1\underline{2}1213) \to (q_2, 12\underline{1}213) \to (q_3, 1\underline{2}2213) \to$
$\to (s, \underline{1}12213) \to (q_1, 1\underline{1}2213) \to (q_1, 11\underline{2}213) \to (q_2, 112\underline{2}13) \to$
$\to (q_2, 1122\underline{1}3) \to (q_3, 112\underline{2}23) \to (s, 11\underline{2}123) \to (q_2, 112\underline{1}23) \to$
$\to (q_3, 11\underline{2}223) \to (s, 1\underline{1}1223) \to (q_1, 11\underline{1}223) \to (q_1, 111\underline{2}23) \to$
$\to (q_2, 1112\underline{2}3) \to (q_2, 11122\underline{3}) \to (q_2, 111222\underline{\#}) \to (q_f, 111222)$

Шукана МТ задається як п'ятірка $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_2, q_3, q_f\}$, $A = \{1, 2, 3, \#\}$, $X = \{1, 2, 3\}$, $Y = \{1, 2\}$, $D = \{\#\}$ і програма роботи δ задана у вигляді таблиці:

δ	1	2	3	#
s	$q_1, 1, R$	$q_2, 2, R$	$q_2, 2, R$	–
q_1	$q_1, 1, R$	$q_2, 2, R$	$q_2, 2, R$	$q_f, \#, L$
q_2	$q_3, 2, L$	$q_2, 2, R$	$q_2, 2, R$	$q_f, \#, L$
q_3	–	$s, 1, L$	–	–

2.8. Приклад. Побудувати машину Тюрінґа в алфавіті $X = \{a, b\}$, яка подвоює у вхідному слові всі входження букви b.

У попередніх прикладах ми бачили, що вставити чи видалити хоча б одну букву зі слова досить непросто. Тому нове слово будемо будувати на вільному місці, а саме, праворуч від вхідного, попередньо дописавши вкінці вхідного слова допоміжний символ =. Для цього в початковому стані s переміщуємося праворуч вкінець слова, міняємо перший порожній символ на = і переходимо в новий стан q_1.



Далі нам потрібно переносити букви за знак =, подвоюючи при цьому букву b. Для цього в стані q_1 „біжимо" на початок слова, переходимо в стан q_2 і запам'ятовуємо першу букву: якщо це буква a, то переходимо в стан q_3, якщо b – в q_4. В станах q_3 і q_4 „біжимо" праворуч до першої порожньої комірки і записуємо в неї першу букву, дублюючи її в стані q_5, якщо цією буквою є b. Потім знову повертаємося ліворуч до тієї букви, яка стала першою у вхідному слові, і повторюємо описані вище дії з цією новою першою буквою.

Даний цикл завершиться тоді, коли в стані q_2 головка МТ буде аналізувати символ =. На останньому етапі витираємо допоміжний символ = і переходимо в кінцевий стан q_f.

Програма роботи δ машини Тюрінґа $MT = (Q, A, \delta, s, q_f)$, де $Q = \{s, q_1, q_2, q_3, q_4, q_5, q_f\}$, $A = \{a, b, =, \#\}$, $X = Y = \{a, b\}$, $D = \{\#, =\}$, задається у вигляді таблиці:

δ	a	b	$=$	$\#$
s	s, a, R	s, b, R	$-$	$q_1, =, L$
q_1	q_1, a, L	q_1, b, L	$q_1, =, L$	$q_2, \#, R$
q_2	$q_3, \#, R$	$q_4, \#, R$	$q_f, \#, R$	$-$
q_3	q_3, a, R	q_3, b, R	$q_3, =, R$	q_1, a, L
q_4	q_4, a, R	q_4, b, R	$q_4, =, R$	q_5, b, R
q_5	$-$	$-$	$-$	q_1, b, L

2.9. Приклад. Побудувати машину Тюрінґа в алфавіті $X = \{a, b\}$, яка подвоює слово, поклавши між ним і його копією знак =.

Ця задача розв'язується аналогічно до попередньої: праворуч від вхідного слова дописуємо символ =, потім повертаємося на початок слова і в циклі копіюємо всі його букви в порожні комірки праворуч.

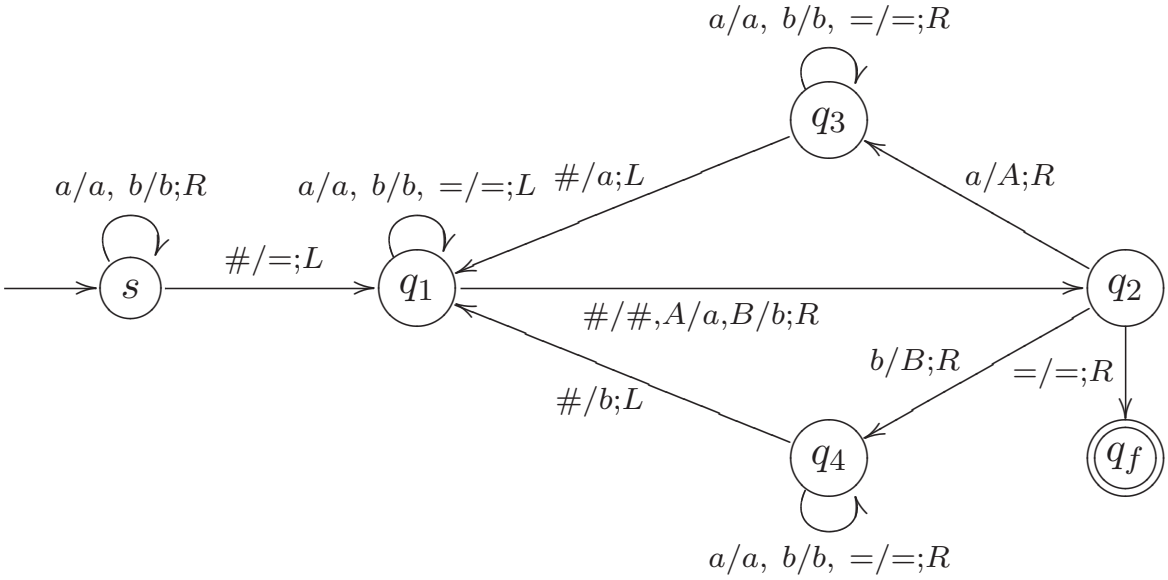
Однак, тут є одна відмінність: букви вхідного слова не видаляються, і це приводить до такої проблеми. Записавши справа копію наступного слова, ми повинні повернутися до вхідного слова в позицію цієї букви і потім перемістити праворуч до наступної букви. Але як дізнатися, в яку позицію вхідного слова потрібно повернутися? В попередній задачі ми завжди поверталися до першої з букв вхідного слова, які залишилися, а тепер ми не видаляємо букви, і тому незрозуміло, які букви ми вже продублювали, а які – ні. Нагадаємо, що в МТ комірки стрічки не номеруються, немає в МТ і лічильників, які б дозволили визначити, скільки букв ми вже продублювали.

В загальному випадку, проблема, з якою ми стикнулись, наступна: як зафіксувати на стрічці деяку позицію, в якій ми вже були і до якої пізніше повинні повернутися? Зазвичай ця проблема вирішується таким чином. Коли ми опиняємося в цій позиції перший раз, то замінюємо активну букву на її двійник – новий допоміжний

символ, причому різні символи замінюємо різними двійниками. Після цього виконуємо дії в інших місцях стрічки. Щоб потім повернутися до нашої позиції, потрібно просто відшукати на стрічці ту комірку, де знаходиться буква-двійник. Потім в активній комірці можна відновити попередню букву.

Таким чином, побудуємо МТ за наступним планом:

- спершу записуємо символ = після останньої букви слова;
- потім повертаємося під першу букву вхідного слова;
- далі замінюємо активну букву на її двійник (a на A, b на B), переміщуємося праворуч до першої вільної комірки і записуємо в неї букву, яку запам'ятали;
- після цього повертаємося ліворуч до комірки з двійником, відновлюємо попередній символ і переміщуємося праворуч до наступної букви;
- аналогічним чином в циклі дублюємо решту букв вхідного слова;
- коли ми продублюємо останню букву вхідного слова і повернемося до її двійника, то в комірці праворуч буде знаходитись символ =. Це ознака того, що вхідне слово повністю продубльоване, і тому роботу МТ потрібно зупинити.



Таким чином, шукана МТ задається як п'ятірка (Q, A, δ, s, q_f), де $Q = \{s, q_1, q_2, q_3, q_4, q_f\}$, $A = \{a, b, A, B, =, \#\}$, $X = \{a, b\}$, $Y = \{a, b, =\}$, $D = \{A, B, \#\}$ і програма роботи δ задана у вигляді таблиці:

δ	a	b	A	B	$=$	$\#$
s	s, a, R	s, b, R	–	–	–	$q_1, =, L$
q_1	q_1, a, L	q_1, b, L	q_2, a, R	q_2, b, R	$q_1, =, L$	$q_2, \#, R$
q_2	q_3, A, R	q_4, B, R	–	–	$q_f, =, R$	–
q_3	q_3, a, R	q_3, a, R	–	–	$q_3, =, R$	q_1, a, L
q_4	q_4, a, R	q_4, b, R	–	–	$q_4, =, R$	q_1, b, L

Рекомендована література : [**7**, с. 183–185], [**19**, с. 7–15], [**21**, с. 163–178].

Питання та вправи до параграфа 2.

В задачах 2.1-2.16 побудувати машину Тюрінґа, яка розпізнає всі слова в алфавіті X і виконує наступну роботу:

2.1. X – український алфавіт. Дописує зліва до вхідного слова ω слово ВУЗ ($\omega \to$ ВУЗω).

2.2. $X = \{$Н, П, У$\}$. Дописує справа до вхідного слова ω слово ПНУ ($\omega \to \omega$ПНУ).

2.3. $X = \{a, b\}$. Залишає в слові тільки першу букву (порожнє слово не міняє).

2.4. $X = \{a, b\}$. Залишає в слові тільки останню букву (порожнє слово не міняє).

2.5. $X = \{a, b, c\}$. Якщо вхідне слово не містить букви c, то замінює всі букви a на b. В іншому випадку – видає слово з однієї букви c.

2.6. $X = \{0, 1, 2, a\}$. З'ясувати, чи є вхідне слово записом числа в трійковій системі числення. Відповідь: 1 – якщо є, 0 – в протилежному випадку.

2.7. $X = \{0, 1, 2\}$. Вважаючи вхідне слово записом числа в трійковій системі числення, видалити з нього всі незначущі нулі.

2.8. $X = \{0, 1, 2\}$. Вважаючи вхідне слово записом числа в трійковій системі числення, з'ясувати, чи ділиться воно на 3. Відповідь: 1 – якщо ділиться, 0 – в протилежному випадку.

2.9. $X = \{a, b\}$. В непорожньому вхідному слові поміняти місцями першу та останню букви.

2.10. $X = \{|\}$. Якщо вхідне слово має непарну довжину, то замінити його на слово $|$, в іншому випадку – на порожнє слово.

2.11. $X = \{a, b, c\}$. Якщо вхідне слово має парну довжину, то залишити в ньому тільки праву половину.

2.12. $X = \{0, 1\}$. Визначити, чи вхідне слово є симетричним (поліндромом), тобто $\omega = \omega^R$. Відповідь: 1 – є, 0 – не є.

2.13. $X = \{a, b\}$. Замінити у вхідному слові кожне входження букви b на ab.

2.14. $X = \{a, b\}$. Видалити із вхідного слова кожне входження підслова ba.

2.15. $X = \{a, b\}$. Подвоїти кожну букву вхідного слова (наприклад, $aba \to aabbaa$).

2.16. $X = \{a, b\}$. Обернути вхідне слово (наприклад, $aab \to baa$).

§ 3. Функції, що обчислюються МТ

В даному параграфі розглядаються функції, для яких існує машина Тюрінґа, що їх обчислює. Дані функції визначаються поведінкою МТ. Аргументи (вхідні слова) з'являються на стрічці до початку обчислення, а значення функції для цього аргументу – слово, яке залишається на стрічці, коли обчислення завершено. Зауважимо, що при аналізі деяких вхідних слів МТ ніколи не переходить у кінцевий стан, і, отже, функція може бути невизначена для деяких вхідних слів-аргументів. В загальному випадку кажемо, що функція $f : (X^*)^n \to Y^*$ є *частково визначеною функцією*, якщо вона не визначена для деяких наборів аргументів $(x_1, \ldots, x_n) \in (X^*)^n$, тобто коли область визначення f є підмножиною $(X^*)^n$. Якщо область визначення f співпадає з $(X^*)^n$, то кажемо, що *функція f повністю визначена на $(X^*)^n$*. Якщо функція f не визначена на наборі $(x_1, \ldots, x_n)$, то писатимемо $f(x_1, \ldots, x_n) = \uparrow$.

Кажемо, що *частково визначена функція* $f : (X^*)^n \to Y^*$ *обчислюється з допомогою машини Тюрінґа*

$$MT = (Q, A = X \cup Y \cup D, \delta, s, q_f),$$

якщо для кожного набору $(x_1, \ldots, x_n) \in (X^*)^n$, на якому f визначена,

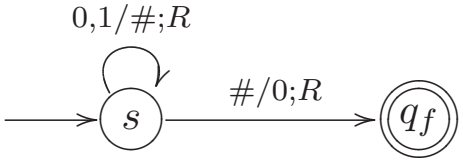
$$(s, x_1 * x_2 * \ldots * x_n) \Rightarrow (q_f, y),$$

де $y = f(x_1, \ldots, x_n) \in Y^*$, а $* \in D$ – спеціальний допоміжний символ, який розділяє вхідні аргументи; і на кожному наборі $(x_1, \ldots, x_n) \in (X^*)^n$, для якого функція не визначена, МТ ніколи не перейде в кінцевий стан q_f.

Надалі найчастіше розглядатимемо числові функції $f : (\mathbb{N}_0)^n \to \mathbb{N}_0$, де $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.

3.1. Приклад. Покажемо, що функція $f : \mathbb{N}_0 \to \mathbb{N}_0$, $f(n) = 0$ обчислюється з допомогою деякої машини Тюрінґа.

Вважатимемо, що невід'ємні цілі числа записані в двійковій системі числення. Для побудови МТ, яка правильно обчислює дану функцію, достатньо визначити стани, в яких МТ замінює всі цифри вхідного слова на порожній символ, друкує цифру 0 і зупиняє роботу. Граф шуканої МТ має вигляд:



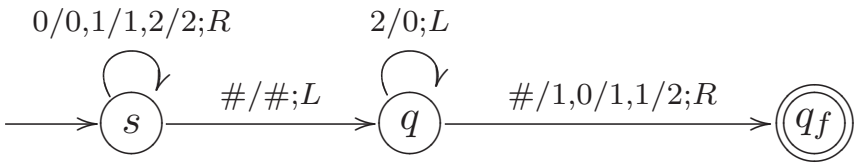
3.2. Приклад. Побудуємо машину Тюрінґа, яка обчислює функцію $f : \mathbb{N} \to \mathbb{N}$, $f(n) = n + 1$ (вважаємо, що натуральні числа задані в трійковій системі числення).

Нехай s – стан, в якому головка МТ рухається в кінець слова. Дійшовши до першої порожньої клітинки, головка МТ переходить ліворуч під останню цифру слова, і керуючий пристрій переходить в стан q. В стані q МТ додає одиницю до тієї цифри, яку вона аналізує в даний момент часу. Спочатку нею є остання цифра числа. Якщо це цифра 0 або 1, то керуючий пристрій замінює її на 1 або 2 відповідно, і МТ зупиняє роботу. Але якщо дана цифра є 2, то керуючий пристрій замінює її на 0, і головка переміщується ліворуч, залишаючись в стані q. Таким чином, тепер МТ додаватиме 1

до попередньої цифри. Якщо нею є цифра 2, то МТ міняє її на 0 і переміщується ліворуч, залишаючись як і до того в стані q, тобто керуючий пристрій повинен виконати ту ж саму дію – збільшити активну цифру на 1. Якщо ж головка перемістилася ліворуч, а в активній комірці немає цифри (а є #), то МТ записує туди 1 і переходить в кінцевий стан q_f.

Зауважимо, що для порожнього вхідного слова наша задача не визначена, тому на цьому слові МТ може поводитися як завгодно. В нашій програмі, наприклад, на порожньому вхідному слові МТ зупиняється і видає 1.

Граф МТ матиме вигляд:



Під унарною системою числення розумітимемо запис невід'ємного цілого числа з допомогою „паличок": має бути виписано стільки паличок, якою є величина числа, наприклад, $2 \to ||$, $5 \to |||||$, $0 \to$ <порожній символ>.

3.3. Приклад. Побудуємо машину Тюрінґа, яка обчислює функцію $f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$, $f(n, m) = n + m$ (вважаємо, що натуральні числа задані в унарній системі числення).

МТ починає аналізувати вхідне слово в конфігурації

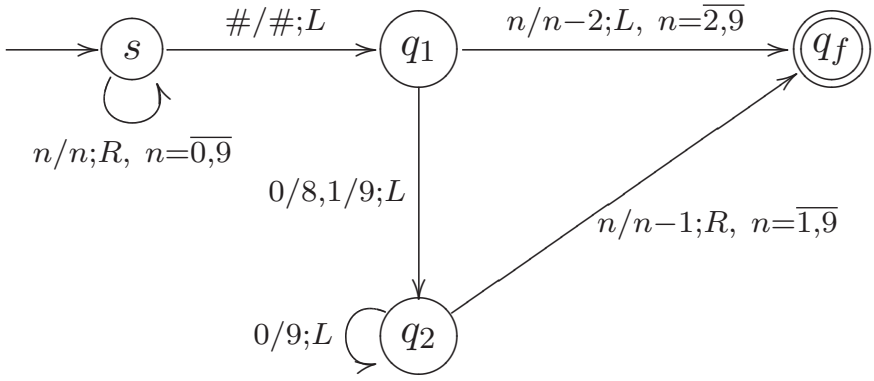
$$(s, \underbrace{||\ldots|}_{n} * \underbrace{||\ldots|}_{m}), \text{ або в конфігурації } (s, \underline{*}\underbrace{||\ldots|}_{m}), \text{ якщо } n = 0.$$

Шукана машина Тюрінґа, яка правильно обчислює дану функцію двох змінних, може працювати, наприклад, таким чином. Якщо $n = 0$, то вона видаляє $*$ і зупиняє роботу. В іншому випадку – видаляє першу паличку $|$, переміщується праворуч до $*$ і на її місці записує $|$. Таким чином $MT = (Q, A = X \cup Y \cup D, \delta, s, q_f)$, де $Q = \{s, q, q_f\}$, $X = Y = \{|\}$, $D = \{*\}$, а програма роботи δ задана у вигляді таблиці:

δ	$\|$	$*$	#
s	$q, \#, R$	$q_f, \#, R$	–
q	$q, \|, R$	$q_f, \|, R$	–

3.4. Приклад. Побудуємо машину Тюрінґа, яка обчислює частково визначену функцію $f : \mathbb{N}_0 \to \mathbb{N}_0,\ f(x) = x - 2$.

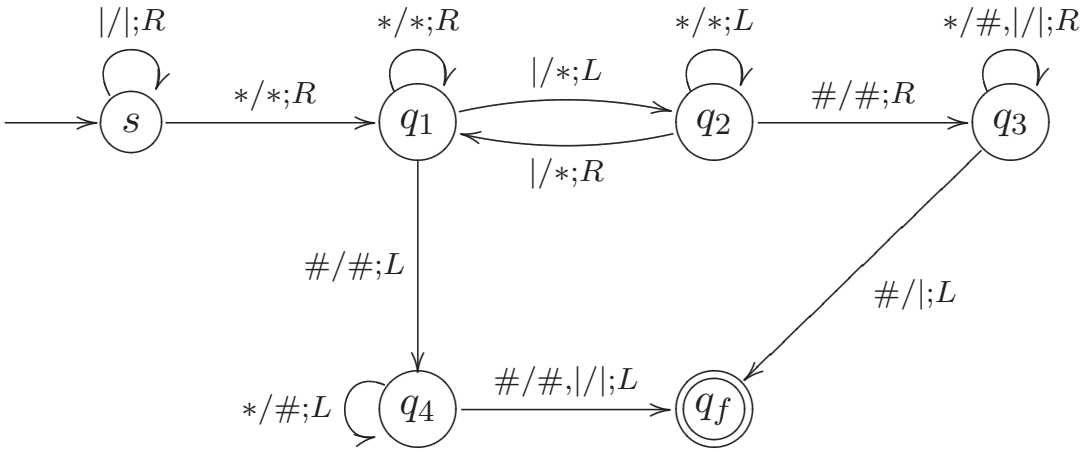
Ідея розв'язання задачі така ж як і в прикладі 3.2. Нехай s – стан, в якому головка МТ рухається вкінець слова. Дійшовши до першої порожньої клітинки головка МТ переходить ліворуч під останню цифру слова, і керуючий пристрій переходить в стан q_1. Якщо в стані q_1, МТ аналізує цифри $2, 3, \ldots, 9$, то вони замінюються на $0, 1, \ldots, 7$ відповідно, і МТ зупиняє роботу. В іншому випадку, якщо це цифри 0 або 1, то керуючий пристрій замінює їх на 8 або 9 відповідно, і головка переміщується ліворуч, керуючий пристрій при цьому переходить в стан q_2, в якому МТ віднімає 1 від попередньої цифри. Якщо ця цифра дорівнює $1, 2, \ldots, 9$, то МТ міняє її на $0, 1, \ldots, 8$ і зупиняє роботу; інакше – замінює 0 на 9 і переміщується ліворуч, залишаючись як і до того в стані q_2, тобто керуючий пристрій повинен виконати ту ж саму дію – зменшити активну цифру на 1 і т.д. Якщо ж головка, перемістившись ліворуч, в стані q_2 аналізує # (це має місце тільки для чисел 0 та 1), то подальші дії МТ не визначені і до таких чисел МТ не застосовна. Побудуємо граф даної МТ:



3.5. Приклад. Побудуємо машину Тюрінґа, яка обчислює функцію $f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0,\ f(n, m) = |n - m|$ (вважаємо, що натуральні числа задані в унарній системі числення).

Нехай початковою конфігурацією є $(s, \underbrace{|\ldots|}_{n} * \underbrace{|\ldots|}_{m})$. Ідея розв'язання задачі така. Спершу МТ переміщується під першу паличку другого числа. Далі порівнює числа наступним чином: у станах q_1 та q_2 МТ відмічає почергово по одній палочці (замінюючи | на $*$) в обох послідовностях | і, вичерпавши одну із послідовностей, робить

висновок, яка з них складається із більшої кількості паличок. Тоді в станах q_3 або q_4 видаляє всі $*$ і на стрічці залишається $n-m$ паличок, якщо $n \geq m$, або $m-n$ паличок, якщо $n < m$. Граф МТ має вигляд:



Проілюструємо роботу МТ на прикладі. Нехай на вході маємо слово $|*|||$. Тоді:

$(s, \underline{|}*|||) \to (s, |\underline{*}|||) \to (q_1, |*\underline{|}||) \to (q_2, |\underline{*}*||) \to (q_2, \underline{|}**||) \to (q_1, **\underline{*}||) \Rightarrow (q_1, ***\underline{|}|) \to (q_2, **\underline{*}*|) \Rightarrow (q_2, \underline{\#}****|) \to (q_3, \underline{*}***|) \Rightarrow (q_3, |\underline{\#}) \to (q_f, \underline{|}|)$.

3.6. Приклад. Побудуємо МТ, яка обчислює функцію $f : \mathbb{N}_0 \times \mathbb{N} \to \mathbb{N}$, $f(n,m) =$ НСД(n,m).

Нехай цілі невід'ємні числа n та m задані в унарній системі числення. Знайдемо їх найбільший спільний натуральний дільник, користуючись алгоритмом Евкліда. Даний алгоритм базується на таких рекурентних формулах для знаходження НСД:

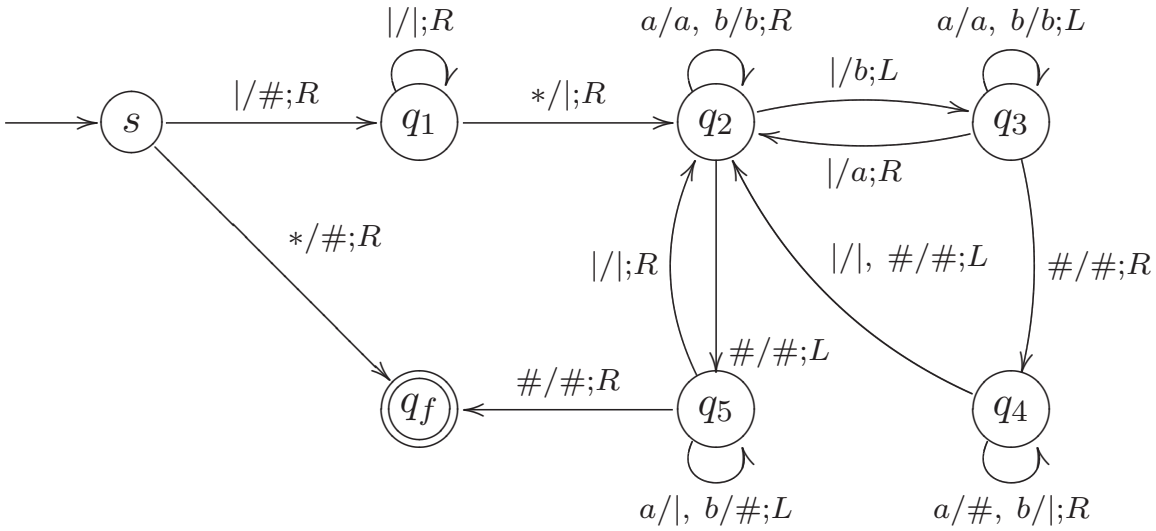
$$\text{НСД}(n,m) = \begin{cases} \text{НСД}(n-m,m), & n > m; \\ \text{НСД}(n,m-n), & n < m; \\ n, & n = m. \end{cases}$$

Побудуємо МТ для відшукання НСД, процес обчислення на якій базуватиметься на почерговому використанні циклів порівняння і віднімання.

Процес побудови МТ будемо супроводжувати ілюстрацією конфігурацій для обчислення НСД(4, 6). Початкова конфігурація матиме вигляд $(s, \underline{|}||| * ||||||)$. Спершу в станах s та q_1 перенесемо першу паличку на місце $*$. В результаті цього отримаємо конфігурацію $(s, |||||\underline{|}||||)$. Далі, використовуючи допоміжні мітки a та b в станах q_2 та q_3, будемо порівнювати числа n і m. При цьому МТ працюватиме так, як би працювала людина, яка порівнює дві довгі послідовності з |, які відразу важко повністю переглянути. Людина відмічала б почергово по одній палочці в обох послідовностях і, вичерпавши одну із послідовностей, зробила б висновок, яка з них складається з більшої кількості паличок. Машина ж замінює першу паличку другого числа на b, потім замінює першу паличку першого числа буквою a, далі знову повертається до паличок другого числа і т.д. Після декількох тактів на стрічці виникає конфігурація $(q_3, \underline{\#}aaaabbbbb|)$, в якій перше число вже вичерпане, а друге – ще ні. Далі МТ переходить в стан q_4, в якому її головка переміщується праворуч, витираючи мітки a та замінюючи мітки b на |. Після цього вона переходить в стан q_2 для порівняння чисел $n = 4$ і $m - n = 2$, знаходячись при цьому в конфігурації $(q_2, |||||\underline{|})$ під першою паличкою числа $m - n = 2$. Таким чином, після циклу порівняння виконано цикл віднімання від другого числа першого, в результаті якого менше число n витерто, а більше число m розбито на n і $m - n$.

Далі знову проходить цикл порівняння, який на цей раз закінчується „вичерпанням" другого числа (що знаходиться праворуч), яке в цьому випадку є меншим. При цьому одержимо конфігурацію $(q_2, ||aabb\underline{\#})$. Наступний такт породжує конфігурацію $(q_5, ||aab\underline{b})$, і починається цикл віднімання від першого числа другого, тобто стирання всіх букв b і заміна букв a палочками. Після цього МТ знову переходить в стан q_2 у конфігурації $(q_2, ||\underline{|}|)$. Цей процес триває доти, доки задачу не буде зведено до випадку двох рівних між собою чисел (в нашому випадку ми вже досягли цього). Насамкінець починається останній цикл порівняння, в якому одне з чисел видаляється, МТ переходить у кінцевий стан q_f, і на стрічці залишається число, записане в унарній системі числення, яке згідно алгоритму Евкліда дорівнює НСД(n, m).

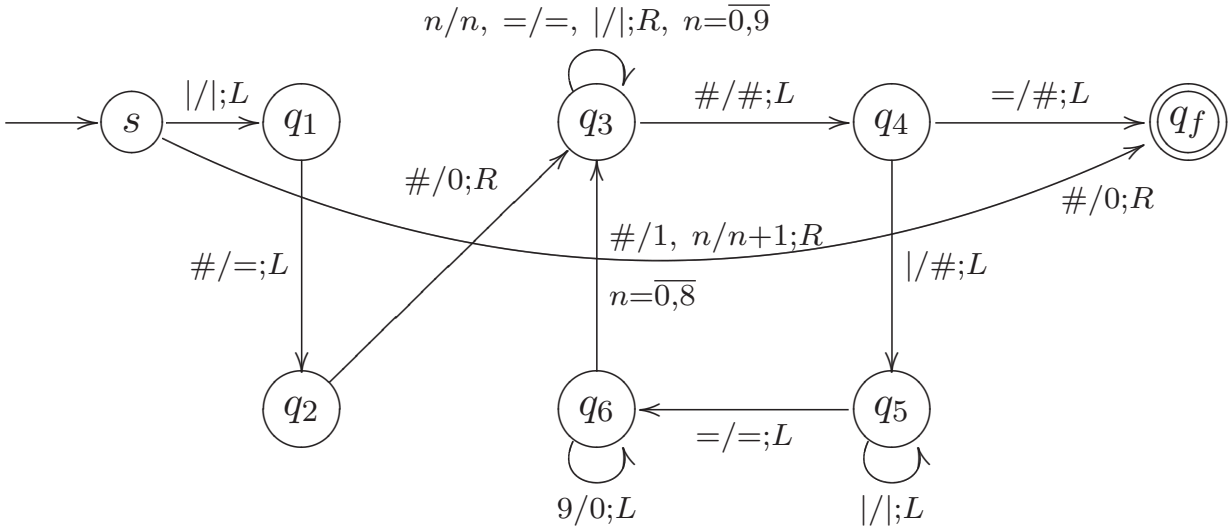
Граф шуканої МТ має вигляд:



Зауважимо, що якщо перше число $n = 0$, то, перебуваючи в конфігурації $(s, \underline{*}|\dots|)$, МТ переходить у кінцевий стан і на виході дає число $m = \text{НСД}(0, m)$, $m \in \mathbb{N}$.

3.7. Приклад. Побудуємо машину Тюрінґа, яка переводить натуральні числа з унарної системи числення в десяткову.

Граф МТ матиме вигляд:



Ідея розв'язання задачі така. Спершу зліва від унарного запису числа, перебуваючи в конфігурації $(s, \underline{|}\dots|)$, в станах q_1 і q_2 дописуємо допоміжний знак $=$ і цифру $\bar{0}$ та переходимо в стан q_3. В результаті цього отримуємо конфігурацію $(q_3, 0\underline{=}|\dots|)$. Далі від унарного числа, розміщеного праворуч, віднімаємо одиницю (видаляємо одну паличку) в стані q_4, переміщуємося ліворуч та в стані

q_6 до десяткового числа додаємо одиницю, діючи за правилами десяткової арифметики. Після цього знову повертаємося до унарного числа і повторюємо циклічно всі операції доти, доки всі палички не будуть витерті. Насамкінець зсуваємося праворуч, в стані q_4 витираємо знак = і зупиняємося.

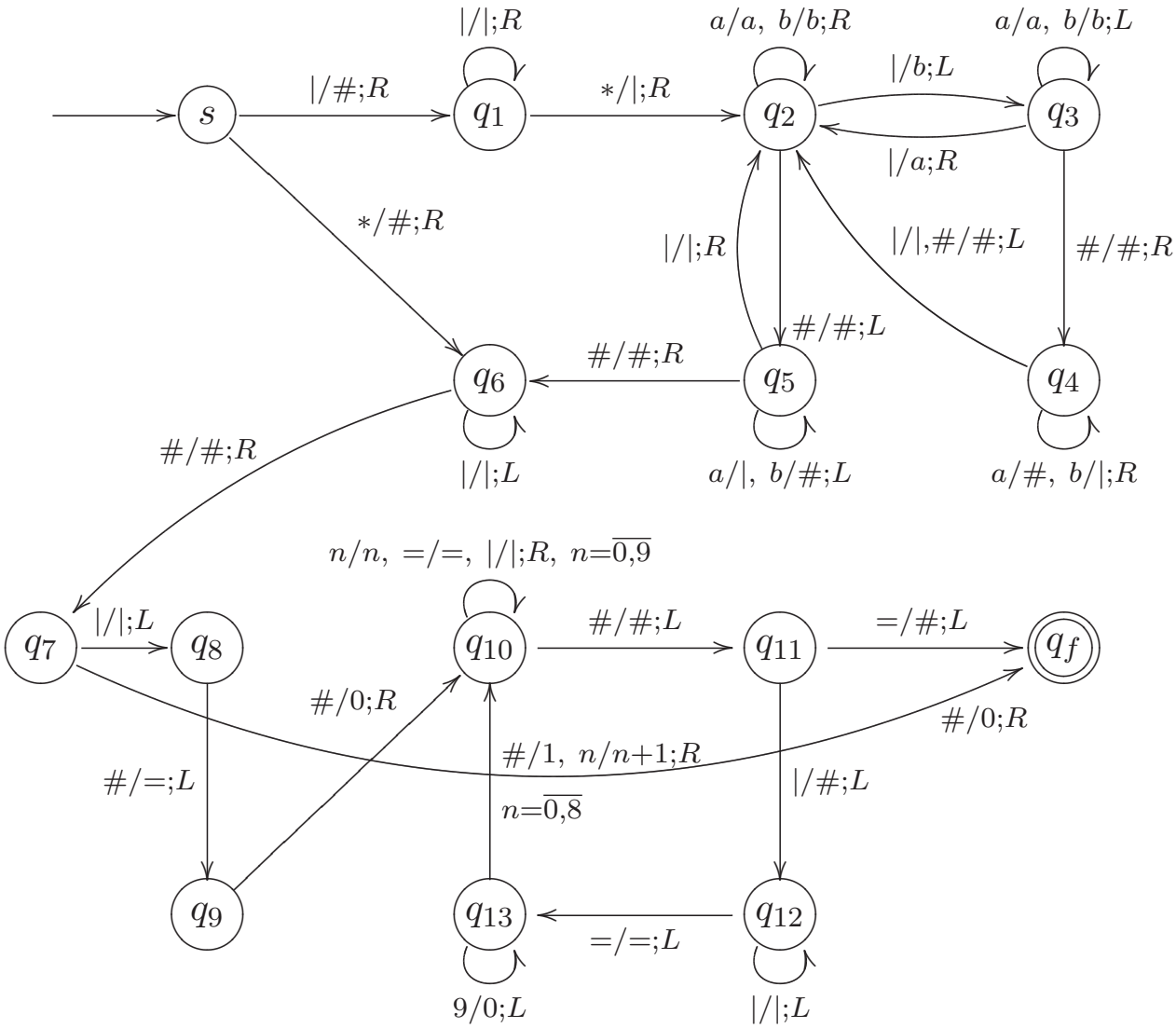
Нехай маємо дві машини Тюрінґа $MT_1 = (Q_1, A_1 = X_1 \cup Y_1 \cup D_1, \delta_1, s_0, s_n)$ і $MT_2 = (Q_2, A_2 = X_2 \cup Y_2 \cup D_2, \delta_2, q_0, q_m)$, причому вважаємо, що вихідний алфавіт першої машини співпадає з вхідним алфавітом другої, тобто $Y_1 = X_2$. Нехай ці МТ обчислюють словесні функції $f_1 : (X_1)^* \to (Y_1)^*$ та $f_2 : (X_2)^* \to (Y_2)^*$ відповідно. Побудуємо МТ, яка обчислює їх композицію $f_2 \circ f_1$. Нагадаємо, що композицією функцій f_1 та f_2 називають таку функцію $f = f_2 \circ f_1$, що для кожного x з області визначення функції f_1 значення $f(x) = f_2(f_1(x))$. При цьому вимагається, щоб функція f_2 була визначена на значенні $f_1(x)$.

Машина Тюрінґа $MT = (Q, A = X \cup Y \cup D, \delta, s, q_f)$ для функції $f = f_2 \circ f_1$ будується наступним чином. У разі потреби стани машини MT_2 перепозначаємо так, щоб вони відрізнялися від станів машини MT_1. Вхідний алфавіт X співпадає із вхідним алфавітом X_1 машини MT_1, вихідний алфавіт Y – з вихідним алфавітом Y_2 машини MT_2, а допоміжний алфавіт $D = X_2 \cup D_1 \cup D_2$. Початковим станом MT є початковий стан s_0 машини MT_1, а кінцевим – кінцевий q_m машини MT_2. Програми роботи δ_1 і δ_2 об'єднуємо і записуємо в одну таблицю, доозначивши, що в стані s_n шукана МТ переміщується ліворуч до початку слова і потім переходить в стан q_0.

Нехай $\omega \in (X_1)^*$. Розглянемо початкову конфігурацію (s, ω). Оскільки $s = s_0$, то на початку роботи MT працює так, як MT_1, і якщо MT_1 застосовна до слова ω, то на деякому кроці буде одержано конфігурацію $(s_n, f_1(\omega))$. Далі в стані s_n головка машини MT переміститься ліворуч під першу букву слова $f_1(\omega)$ і керуючий пристрій перейде в стан q_0, тобто одержимо конфігурацію $(q_0, f_1(\omega))$. Якщо MT_2 застосовна до слова $f_1(\omega)$, то на деякому кроці отримаємо конфігурацію $(q_m, f_2(f_1(\omega)))$, яка є заключною для MT, бо $q_f = q_m$. Таким чином, якщо MT_1 застосовна до ω і MT_2 застосовна до $f_1(\omega)$, то і MT застосовна до ω. Машина MT називається композицією машин MT_1 та MT_2 і позначається через $MT_2 \circ MT_1$.

Означення композиції залишається без зміни, якщо MT_1 та MT_2 обчислюють функції кількох змінних. Важливо лише, щоб дані для MT_2 були у відповідному вигляді підготовлені машиною MT_1.

3.8. Приклад. Розглянемо МТ, задану графом:



Дана МТ обчислює функцію $f : \mathbb{N}_0 \times \mathbb{N} \to \mathbb{N}$, $f(n, m) =$ НСД(n, m) і видає результат в десятковій системі числення. Вона є композицією МТ, побудованих в прикладах 3.6 та 3.7.

Аналізуючи попередні приклади, у читача складається враження, що процес, який відбувається на МТ, є сповільненим переглядом процесу обчислення, який виконується людиною відповідно до деякого алгоритму. Разом з тим розглянуті приклади наштовхують на думку, що з допомогою МТ можна задавати й інші відомі нам алгоритми. Цілком природно виникає питання: чи спосіб задання

алгоритмів з допомогою МТ є універсальним в тому сенсі, що будь-який алгоритм можна подати таким чином? На це питання сучасна теорія алгоритмів дає відповідь з допомогою наступної гіпотези:

Теза Тюрінґа-Черча. *Будь-який алгоритм можна реалізувати на відповідній машині Тюрінґа.*

Перш за все звернемо увагу на наступну характерну особливість даної гіпотези. В її формулюванні мова йде, з одного боку, про загальне поняття алгоритму, яке не є точним математичним поняттям; з другого боку, в цьому ж формулюванні говориться про таке точне математичне поняття як машина Тюрінґа. Тому не може йти і мови про доведення даної гіпотези подібно до того, як доводяться теореми в математиці. Значення гіпотези полягає саме в тому, що вона уточнює загальне, але розпливчасте поняття „будь-якого алгоритму" з допомогою цілком точного математичного поняття МТ. Таким чином, теорія алгоритмів оголошує об'єктом своїх досліджень машини Тюрінґа. Тепер вже стають коректними питання можливості чи неможливості побудови алгоритму для задачі того чи іншого типу.

Справедливість гіпотези підтверджується практикою, яка, як відомо з філософії, є єдиним критерієм істини. Всі відомі алгоритми, які були придумані впродовж багатьох тисячоліть історії математики, можуть бути задані з допомогою МТ.

Рекомендована література : [**1**, с. 33–45], [**7**, с. 186–194], [**10**, с. 324–333], [**16**, с. 319–323], [**23**, с. 83–86].

Питання та вправи до параграфа 3.

3.1. Дайте означення частково визначеної функції, що обчислюється МТ.

3.2. Що ви розумієте під композицією МТ?

3.3. Побудувати машину Тюрінґа, що обчислює числову функцію $f : \mathbb{N}_0 \to \mathbb{N}_0$, аргументи якої задані в трійковій системі числення.

б) $f(x) = 0$;

а) $f(x) = \begin{cases} 0, & x = 0 \\ 1, & x \neq 0; \end{cases}$

б) $f(x) = x + 2$;

в) $f(x) = x - 2$;
г) $f(x) = 2 \cdot x + 1$;
д) $f(x) = 9_{10} \cdot x + 2$.

3.4. Побудувати машину Тюрінґа, яка обчислює числову функцію $P_i^4 : (N_0)^4 \to \mathbb{N}_0$, $P_i^4(x_1, x_2, x_3, x_4) = x_i$, де $i = 1, 2, 3, 4$ (вважаємо, що аргументи x_k задані в унарній системі числення).

3.5. Побудувати машину Тюрінґа, що обчислює частково визначену числову функцію $f : (\mathbb{N}_0)^2 \to \mathbb{N}_0$, аргументи якої задані в унарній системі числення.

а) $f(x, y) = \begin{cases} x - y, & x > y \\ 0, & x \le y; \end{cases}$

б) $f(x, y) = \begin{cases} x - y, & x \ge y \\ \uparrow, & x < y; \end{cases}$

в) $f(x, y) = xy$;
г) $f(x, y) = \min\{x, y\}$;
д) $f(x, y) = \max\{x, y\}$;
е) $f(x, y) = [\frac{x}{y}]$;
є) $f(x, y) = x - y[\frac{x}{y}]$.

3.6. Побудувати машину Тюрінґа, що обчислює частково визначену числову функцію $f : \mathbb{N}_0 \to \mathbb{N}_0$, аргументи якої задані в унарній системі числення.

а) $f(x) = \begin{cases} \frac{x}{2}, & x - \text{ парне} \\ \uparrow, & x - \text{ непарне}; \end{cases}$

б) $f(x) = [\frac{x}{2}]$.

3.7. Побудувати машину Тюрінґа, яка обчислює частково визначену числову функцію $f : (\mathbb{N}_0)^2 \to \mathbb{N}_0$.

а) $f(x, y) = x + y$;
б) $f(x, y) = x - y$.

Вважаємо, що аргумент x заданий в четвірковій системі числення, а y – в трійковій системі. Результат отримати в четвірковій системі числення.

3.8. Доведіть, що функція

$$f_p(x) = \begin{cases} 1, & \text{якщо } x \text{ ділиться на } p \\ 0, & \text{якщо } x \text{ не ділиться на } p \end{cases}$$

обчислюється деякою МТ.

3.9. Побудувати машину Тюрінґа, яка обчислює словесну функцію $f : \{a, b\}^* \to \{a, b\}^*$.

а) $f(\omega) = \omega ab$;
б) $f(\omega) = \omega^R$;
в) $f(\omega) = \omega\omega$;
г) $f(\omega) = \omega\omega^R$.

3.10. Побудувати машину Тюрінґа, яка переводить натуральні числа з трійкової системи числення в унарну.

3.11. Побудувати машину Тюрінґа, яка переводить натуральні числа з четвіркової системи числення в двійкову.

3.12. Побудувати машину Тюрінґа, яка переводить натуральні числа з двійкової системи числення в четвіркову (підказка: врахуйте, що в двійковому числі може бути непарна кількість букв).

3.13. Скориставшись композицією МТ, побудуйте машину Тюрінґа, яка за записом числа в унарній системі числення визначає, чи є це число степенем 3 $(1, 3, 9, 27, \ldots)$. Відповідь: 1 якщо так, 0 – в протилежному випадку.

§ 4. Рекурсивні функції

Будь-який алгоритм співставляє допустимим вхідним даним результат. Це означає, що з кожним алгоритмом однозначно зв'язана функція, яку він обчислює. В цьому параграфі розглядатимуться алгоритмічно обчислювані функції. Ми побудуємо клас всіх функцій такого виду з допомогою тільки трьох найпростіших функцій і трьох операцій над такими функціями. Даний підхід до формалізації поняття алгоритму належить Геделю. Основна ідея Геделя полягала в тому, щоб одержати всі обчислювані функції із невеликої кількості базових функцій з допомогою найпростіших алгоритмічних засобів.

Назвемо найпростішими такі функції:

- нуль-функцію: $O(x) = 0$, $x \in \mathbb{N}_0 = \mathbb{N} \cup \{0\}$;
- функцію наступності: $S(x) = x + 1$, $x \in \mathbb{N}_0$;
- функції проектування: $P_i^n(x_1, x_2, \ldots, x_i, \ldots x_n) = x_i$, $x_i \in \mathbb{N}_0$.

До даних функцій будемо застосовувати наступні операції.

Композиція. Нехай $m, k \in \mathbb{N}$ і задано числові функції $g : \mathbb{N}_0^m \to \mathbb{N}_0$ і $h_i : \mathbb{N}_0^k \to \mathbb{N}_0$, де $i = 1, 2, \ldots, m$. Визначимо функцію $f : \mathbb{N}_0^k \to \mathbb{N}_0$ за правилом

$$f(n_1, \ldots, n_k) = g(h_1(n_1, \ldots, n_k), \ldots, h_m(n_1, \ldots, n_k)).$$

Кажемо, що *функція f одержана з допомогою операції композиції з функцій g і $h_1, \ldots, h_m$*. Функцію f позначимо через $g \circ (h_1, \ldots, h_m)$.

Примітивна рекурсія. Нехай $k \in \mathbb{N}_0$ і задано числові функції $g : \mathbb{N}_0^k \to \mathbb{N}_0$ і $h : \mathbb{N}_0^{k+2} \to \mathbb{N}_0$ (у випадку, коли $k = 0$, функція g ототожнюється з числом із множини $\mathbb{N}_0$). Визначимо функцію $f : \mathbb{N}_0^{k+1} \to \mathbb{N}_0$ за правилом

$$f(n_1, \ldots, n_k, 0) = g(n_1, \ldots, n_k),$$

$$f(n_1, \ldots, n_k, m + 1) = h(n_1, \ldots, n_k, m, f(n_1, \ldots, n_k, m)), m \geq 0.$$

В цьому випадку кажемо, що *функція f отримана з функцій g і h з допомогою операції примітивної рекурсії*.

Клас примітивно рекурсивних функцій визначається наступним чином:

- найпростіші функції O, S і P_i^n є примітивно рекурсивними функціями;
- якщо $g : \mathbb{N}_0^m \to \mathbb{N}_0$ і $h_i : \mathbb{N}_0^k \to \mathbb{N}_0$, де $i = 1, 2, \ldots, m$, є примітивно рекурсивними функціями, то $g \circ (h_1, \ldots, h_m)$ також є примітивно рекурсивною функцією;
- якщо $g : \mathbb{N}_0^k \to \mathbb{N}_0$ і $h : \mathbb{N}_0^{k+2} \to \mathbb{N}_0$ є примітивно рекурсивними функціями, то функція f, одержана з g і h за допомогою операції примітивної рекурсії, також є примітивно рекурсивною;
- інших примітивно рекурсивних функцій не існує.

Іншими словами, функцію називають *примітивно рекурсивною*, якщо її можна одержати з найпростіших функцій за допомогою

скінченної кількості застосувань операцій композиції та примітивної рекурсії.

4.1. Приклад. Довести, що функція $O_k(n_1, n_2, \ldots, n_k) = 0$ є примітивно рекурсивною.

Дану функцію можна зобразити у вигляді композиції двох найпростіших примітивно рекурсивних функцій. Дійсно,

$$O_k(n_1, n_2, \ldots, n_k) = O(P_1^k(n_1, n_2, \ldots, n_k)).$$

Таким чином, функція $O_k = O \circ P_1^k$ є примітивно рекурсивною.

4.2. Приклад. Показати, що функції $S_i^k(n_1, \ldots, n_i, \ldots, n_k) = n_i + 1$, де $i = 1, \ldots, k$, є примітивно рекурсивними.

Функції S_i^k можна визначити як

$$S_i^k(n_1, n_2, \ldots, n_k) = S(P_i^k(n_1, \ldots, n_i, \ldots, n_k)).$$

Отже, функція $S_i^k = S \circ P_i^k$ є примітивно рекурсивною, бо вона одержана з примітивно рекурсивних функцій з допомогою операції композиції.

4.3. Приклад. Довести, що сталі функції $C_a^k(n_1, n_2, \ldots, n_k) = a$, де $k, a \in \mathbb{N}_0$, є примітивно рекурсивними.

Дані функції можна зобразити у вигляді скінченної кількості композицій примітивно рекурсивних функцій

$$C_a^k(n_1, n_2, \ldots, n_k) = \underbrace{S(S \ldots (S}_{a}(O_k(n_1, n_2, \ldots, n_k))) \ldots).$$

4.4. Приклад. Показати, що функція $Suma : \mathbb{N}_0^2 \to \mathbb{N}_0$, $Suma(n, m) = n + m$ є примітивно рекурсивною.

Функцію $Suma$ можна визначити наступним чином.

$$Suma(n, 0) = n = P_1^1(n),$$

$$Suma(n, m+1) = n+(m+1) = (n+m)+1 = S(P_3^3(n, m, Suma(n, m))).$$

Отже, $Suma$ є примітивно рекурсивною функцією, оскільки її одержано з примітивно рекурсивних функцій $P_1^1 : \mathbb{N}_0 \to \mathbb{N}_0$ і $S \circ P_3^3 : \mathbb{N}_0^3 \to \mathbb{N}_0$ з допомогою операції примітивної рекурсії.

4.5. Приклад. Довести, що функція $Prod : \mathbb{N}_0^2 \to \mathbb{N}_0$, $Prod(n, m) = nm$ є примітивно рекурсивною.

Функцію $Prod$ можна одержати з примітивно рекурсивних функцій O і $Suma \circ (P_3^3, P_1^3)$ з допомогою операції примітивної рекурсії наступним чином.

$$Prod(n,0) = 0 = O(n),$$
$$Prod(n,m+1) = n(m+1) = nm + n =$$
$$= Suma(P_3^3(n,m,Prod(n,m)), P_1^3(n,m,Prod(n,m))).$$

4.6. Приклад. Показати, що функція $Exp(n,m) = n^m$ є примітивно рекурсивною.

Функцію Exp можна визначити як

$$Exp(n,0) = 1 = S(O(n)),$$
$$Exp(n,m+1) = n^{m+1} = n^m n =$$
$$= Prod(P_3^3(n,m,Exp(n,m)), P_1^3(n,m,Exp(n,m))).$$

4.7. Приклад. Довести, що функції

а) $Prec : \mathbb{N}_0 \to \mathbb{N}_0, \quad Prec(n) = n \div 1 = \begin{cases} 0, & n = 0 \\ n-1, & n > 0 \end{cases}$

б) $Minus : \mathbb{N}_0^2 \to \mathbb{N}_0, \quad Minus(n,m) = n \div m = \begin{cases} 0, & n \le m \\ n-m, & n > m \end{cases}$

в) $Abs(n,m) = |n-m|$ є примітивно рекурсивними.

а) Функцію $Prec$ можна одержати з константи і проектування $P_1^2 : \mathbb{N}_0^2 \to \mathbb{N}_0$ з допомогою операції примітивної рекурсії.

$$Prec(0) = 0,$$
$$Prec(m+1) = m = P_1^2(m, Prec(m)).$$

б) Функцію $Minus$ можна одержати з допомогою функції $Prec$ наступним чином.

$$Minus(n,0) = n = P_1^1(n),$$
$$Minus(n,m+1) = n \div (m+1) = (n \div m) \div 1 =$$
$$= Prec(P_3^3(n,m,Minus(n,m))).$$

в) $Abs(n,m) = |n-m| = (n \div m) + (m \div n) =$
$$= Suma(Minus(n,m), Minus(m,n)).$$

Оскільки метою цього розділу є показати, що клас примітивно рекурсивних функцій є досить широким, то слід довести, що булеві функції, які використовуються в мовах програмування високого рівня, є примітивно рекурсивними. В наступних вправах через 1 позначаємо значення „істина”, через 0 – „хибність”.

4.8. Приклад. Показати, що наступні функції є примітивно рекурсивними.

а) $Neg(n) = \begin{cases} 0, n \geq 1 \\ 1, n = 0 \end{cases}$

б) $And(n,m) = \begin{cases} 1, n \geq 1 \text{ і } m \geq 1 \\ 0, \text{ в протилежному випадку} \end{cases}$

в) $Or(n,m) = \begin{cases} 1, n \geq 1 \text{ або } m \geq 1 \\ 0, \text{ в протилежному випадку} \end{cases}$

г) $If\text{-}then\text{-}else(n,m,k) = \begin{cases} m, n \geq 1 \\ k, \text{ в протилежному випадку.} \end{cases}$

Дійсно, дані функції утворюються з примітивно рекурсивних функцій наступним чином:

а) $Neg(n) = Minus(1,n)$;
б) $And(n,m) = Neg(Neg(Prod(n,m)))$;
в) $Or(n,m) = Neg(And(Neg(n), Neg(m)))$;
г) $If\text{-}then\text{-}else(n,m,k) =$

$$= Suma(Prod(Neg(Neg(n)), m), Prod(Neg(n), k))).$$

Надалі писатимемо „x *and* y”, „x *or* y” та „if x then y else z” замість $And(x,y)$, $Or(x,y)$ та $If\text{-}then\text{-}else(x,y,z)$ відповідно.

4.9. Приклад. Довести, що наступні функції є примітивно рекурсивними.

а) $Eq(n,m) = \begin{cases} 1, n = m \\ 0, n \neq m \end{cases}$

б) $Gr(n,m) = \begin{cases} 1, n > m \\ 0, n \leq m \end{cases}$

в) $Geq(n,m) = \begin{cases} 1, n \geq m \\ 0, n < m \end{cases}$

г) $Ls(n,m) = \begin{cases} 1, n < m \\ 0, n \geq m \end{cases}$

д) $Leq(n,m) = \begin{cases} 1, n \leq m \\ 0, n > m \end{cases}$

Дані функції утворюються з примітивно рекурсивних функцій так:

а) $Eq(n,m) = Neg(Add(Minus(n,m), Minus(m,n)))$;
б) $Gr(n,m) = Neg(Neg(Minus(n,m)))$;
в) $Geq(n,m) = Gr(n,m)$ or $Eq(n,m)$;
г) $Ls(n,m) = Neg(Geq(n,m))$;
д) $Leq(n,m) = Neg(Gr(n,m))$.

4.10. Приклад. Довести, що для кожного $k \in \mathbb{N}$ функція

$$Max^k(n_1, n_2, \ldots, n_k) = \max\{n_1, n_2, \ldots, n_k\}$$

є примітивно рекурсивною.

Доведемо дане твердження індукцією по k. Для $k = 1$ твердження вірне, бо $Max^1(n_1) = P_1^1(n_1)$. Припустимо, що твердження виконується для $k-1$ і доведемо для k:
$Max^k(n_1, \ldots, n_k) =$
$=$ if $n_k >$ $Max^{k-1}(n_1, \ldots, n_{k-1})$ then n_k else $Max^{k-1}(n_1, \ldots, n_{k-1})$.

4.11. Приклад. Застосувати операцію примітивної рекурсії до примітивно рекурсивних функцій $g : \mathbb{N}_0 \to \mathbb{N}_0$ і $h : \mathbb{N}_0^3 \to \mathbb{N}_0$.

а) $g(n) = 3^n$, $h(n_1, n_2, n_3) = 4n_2 + n_3$;
б) $g(n) = 2^n$, $h(n_1, n_2, n_3) = 3^{n_1} n_3^{n_1}$ (вважаємо, що $0^0 = 1$).

а) $f(n,0) = g(n) = 3^n$
$f(n, m+1) = h(n, m, f(n,m)) = 4m + f(n,m)$
$f(n,1) = 4 \cdot 0 + f(n,0) = 3^n$
$f(n,2) = 4 \cdot 1 + f(n,1) = 4 + 3^n$
$f(n,3) = 4 \cdot 2 + f(n,2) = 4 \cdot 2 + 4 \cdot 1 + 3^n$
Доведемо індукцією по m, що
$f(n,m) = 4((m-1) + \ldots + 2 + 1) + 3^n = 4\frac{(m-1)+1}{2}(m-1) + 3^n =$
$= 2m(m-1) + 3^n$.

Припустимо, що твердження виконується для $m = k$, тобто $f(n,k) = 2k(k-1) + 3^n$.

Для $m = k + 1$ маємо $f(n, k + 1) = 4k + f(n, k) = 4k + 2k(k - 1) + 3^n = 2k(2 + k - 1) + 3^n = 2(k + 1)k + 3^n$.

Таким чином, для будь-яких $n, m \in \mathbb{N}_0$

$$f(n, m) = 2m(m - 1) + 3^n$$

Функція f отримана з примітивно рекурсивних функцій g і h з допомогою операції примітивної рекурсії і, отже, є примітивно рекурсивною.

б) $f(n, 0) = g(n) = 2^n$

$f(n, m + 1) = h(n, m, f(n, m)) = 3^n(f(n, m))^n$

$f(n, 1) = 3^n(f(n, 0))^n = 3^n 2^{n^2}$

$f(n, 2) = 3^n(3^n 2^{n^2})^n = 3^n 3^{n^2} 2^{n^3} = 3^{n+n^2} 2^{n^3}$

$f(n, 3) = 3^n(3^{n+n^2} 2^{n^3})^n = 3^{n+n^2+n^3} 2^{n^4}$

Доведемо індукцією по m, що

$f(n, m) = 3^{n+n^2+\ldots n^m} 2^{n^{m+1}} = 3^{\frac{n(n^m-1)}{n-1}} 2^{n^{m+1}}$.

Припустимо, що твердження виконується для $m = k$, тобто $f(n, k) = 3^{\frac{n(n^k-1)}{n-1}} 2^{n^{k+1}}$.

Для $m = k + 1$ маємо

$f(n, k+1) = 3^n(f(n, k))^n = 3^n(3^{\frac{n(n^k-1)}{n-1}} 2^{n^{k+1}})^n = 3^n 3^{\frac{n^2(n^k-1)}{n-1}} 2^{n^{k+2}}$
$= 3^{n+\frac{n^2(n^k-1)}{n-1}} 2^{n^{k+2}} = 3^{\frac{n^2-n+n^{k+2}-n^2}{n-1}} 2^{n^{k+2}} = = 3^{\frac{n(n^{k+1}-1)}{n-1}} 2^{n^{k+2}}$.

Таким чином, для будь-яких $n, m \in \mathbb{N}_0$

$$f(n, m) = 3^{\frac{n(n^m-1)}{n-1}} 2^{n^{m+1}}.$$

Дана функція f отримана з примітивно рекурсивних функцій g та h, і тому є примітивно рекурсивною.

Мінімізація. Розглянемо частково визначену функцію n змінних $f(x_1, \ldots, x_n)$. Визначимо функцію $g(x_1, \ldots, x_n)$ наступним чином. Зафіксуємо певні $n - 1$ значення аргументів $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots x_n$ функції f та розглянемо рівняння $f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n) = x_i$. Обчислюватимемо значення $f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n)$ для $y = 0, 1, 2, \ldots$.

Розглянемо наступні випадки:

- значення $f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$ не визначено;

- значення $f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n)$ для $y = 0, 1, \ldots, a-1$ визначені, але вони відмінні від x_i, а значення $f(x_1, \ldots, x_{i-1}, a, x_{i+1}, \ldots, x_n)$ не визначено;
- значення $f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n)$ визначені для всіх $y \in \mathbb{N}_0$, але вони відмінні від x_i.

В усіх цих випадках значення функції $g(x_1, \ldots, x_n)$ вважають не визначеним. Інакше, нехай

$$a = \min\{y \in \mathbb{N}_0 \mid f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n) = x_i\}.$$

У цьому випадку зазначений процес зупиняється і дає найменший розв'язок $y = a$ рівняння $f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n) = x_i$, який є значенням функції $g(x_1, \ldots, x_n) = a$. Значення функції g для заданої функції f залежить від значень параметрів $x_1, \ldots, x_n$, тому функція g є частково визначеною функцією змінних $x_1, \ldots, x_n$. Кажуть, що функція g отримана з функції f з допомогою *операції мінімізації за змінною* x_i.

Функцію f називають *частково рекурсивною*, якщо вона може бути утворена з найпростіших функцій за допомогою скінченної кількості застосувань операцій композиції, примітивної рекурсії та мінімізації. Всюди визначені частково рекурсивні функції називаються *загальнорекурсивними*.

4.12. Приклад. Розглянемо функцію $f : \mathbb{N}_0 \to \mathbb{N}_0$, $f(x) = x + 2$. За допомогою операції мінімізації визначимо функцію $g : \mathbb{N}_0 \to \mathbb{N}_0$. Для цього розглянемо рівняння $y + 2 = x$. Бачимо, що функція g не визначена, якщо $x = 0$ або $x = 1$, і $g(x) = x - 2$, якщо $x \geq 2$.

4.13. Приклад. Доведемо, що частково визначена функція $g : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$,

$$g(x_1, x_2) = \frac{x_2}{1 - x_1 x_2^{2012}}$$

є частково рекурсивною. Застосуємо операцію мінімізації за змінною x_2 до примітивно рекурсивної функції $f(x_1, x_2) = (1 \div x_1)x_2$. В результаті отримаємо рівняння $(1 \div x_1)y = x_2$. Отже,

$$g(x_1, x_2) = \begin{cases} x_2, & x_1 = 0, \\ 0, & x_2 = 0, \\ \uparrow, & x_1 \neq 0 \text{ і } x_2 \neq 0, \end{cases} \qquad = \frac{x_2}{1 - x_1 x_2^{2012}}.$$

В даному параграфі ми показали, що клас частково рекурсивних функцій є досить широким. Поняття частково рекурсивної функції – одне з основних понять теорії алгоритмів. Його значення є таким. З одного боку, кожна стандартно задана частково рекурсивна функція є обчислювана за певною процедурою, яка відповідає інтуїтивному уявленню алгоритму, а з іншого – які б досі не будувалися класи точно визначених алгоритмів, завжди з'ясовувалося, що числові функції, які обчислювалися за алгоритмами цих класів, були частково рекурсивними. Тому загальноприйнятою є така наукова гіпотеза:

Теза Черча. *Клас алгоритмічно обчислюваних частково визначених числових функцій збігається з класом усіх частково рекурсивних функцій.*

У формулювання цієї тези входить інтуїтивне поняття обчислюваності, тому її не можна ні довести, ні спростувати в загальноматематичному значенні. Це факт, на користь якого свідчить багаторічна математична практика. Проте справедливою є наступна теорема, доведення якої читач може знайти в [**10**]:

4.14. Теорема. *Функція є обчислюваною за Тюрінґом тоді і тільки тоді, коли вона є частково рекурсивною.*

Рекомендована література : [**7**, с. 195–202], [**10**, с. 333–354], [**13**], [**16**, с. 323–327], [**20**, с. 194–207].

Питання та вправи до параграфа 4.

4.1. Дайте означення примітивно рекурсивної функції.

4.2. Опишіть операцію мінімізації частково рекурсивної функції.

4.3. Як пов'язнані частково рекурсивні функції і машини Тюрінґа?

4.4. Довести, що наступні функції однієї змінної є примітивно рекурсивними:

а) $f(n) = 3n + 4$;
б) $f(n) = 2^n + n^2$;
в) $f(n) = n!$;

г) $sg(n) = \begin{cases} 0, & n = 0 \\ 1, & n > 0. \end{cases}$

4.5. Довести, що наступні функції $f : \mathbb{N}_0^2 \to \mathbb{N}_0$ є примітивно рекурсивними:

а) $f(n, m) = \min\{n, m\}$;
б) $f(n, m) = \max\{n, m\}$.

4.6. Застосувати операцію примітивної рекурсії до функцій $g : \mathbb{N}_0 \to \mathbb{N}_0$ і $h : \mathbb{N}_0^3 \to \mathbb{N}_0$. Записати результуючу функцію $f : \mathbb{N}_0^2 \to \mathbb{N}_0$ аналітично.

а) $g(n) = 2^n$, $h(n_1, n_2, n_3) = 3n_2 + n_3$;
б) $g(n) = 2012n^{2012}$, $h(n_1, n_2, n_3) = 5n_2 + n_3$;
в) $g(n) = 3^n$, $h(n_1, n_2, n_3) = 2n_1 + 5n_3$;
г) $g(n) = 2012^n$, $h(n_1, n_2, n_3) = 5^{n_1} n_3^{n_1}$ (вважаємо, що $0^0 = 1$);
д) $g(n) = 2^n + n^2$, $h(n_1, n_2, n_3) = n_1 + 3n_3$;
е) $g(n) = 1$, $h(n_1, n_2, n_3) = n_3(sg|n_1 + 2 - 2n_3|)$.

4.7. Застосувати операцію мінімізації до функції f за всіма її змінними. Подати вихідні функції g аналітично.

а) $f(n) = 2$;
б) $f(n) = [\frac{n}{3}]$;
в) $f(n, m) = n + m$;
г) $f(n, m) = 2 - n - m$;
д) $f(n, m) = P_1^2(n, m)$;
е) $f(n, m) = \max\{n, m\}$;
є) $f(n, m) = \min\{n, m\}$;
ж) $f(n, m) = n \div m$.

4.8. Застосувавши операцію мінімізації до відповідної примітивно рекурсивної функції f, довести, що функція g є частково рекурсивною.

а) $g(n) = 3 - n$;
б) $g(n) = \frac{n}{3}$;
в) $g(n, m) = n - 2m$;
г) $g(n) = \frac{1}{n+1}$;
д) $g(n, m) = \frac{n}{m+2}$.

4.9. Довести, що застосувавши один раз операцію мінімізації до всюди визначеної числової функції, отримаємо функцію, яка визначена принаймні в одній точці.

4.10. Навести приклад функції $f : \mathbb{N}_0 \to \mathbb{N}_0$, застосувавши до якої операцію мінімізації двічі, отримаємо ніде не визначену функцію.

4.11. Сформулювати необхідну і достатню умову того, що функція g, отримана з функції f з допомогою операції мінімізації, є

а) всюдивизначеною;
б) ніде не визначеною.

§ 5. Нормальні алгоритми Маркова

В даному параграфі розглядається ще один важливий підхід до уточнення поняття алгоритму, розроблений А.А. Марковим. Нормальні алгоритми Маркова (НАМ) базуються на перетворенні слів в деякому скінченному алфавіті в відповідності з певним чином визначеними правилами. В НАМ використовується тільки одна елементарна дія – підстановка, яка визначається наступними чином. Нехай $A = \{a_1, a_2, \ldots, a_n\}$ – алфавіт. Якщо α і β – слова в алфавіті A, то вирази $\alpha \to \beta$ і $\alpha \mapsto \beta$ називаються *формулами підстановки в алфавіті A* (вважаємо, що символи $\to$ та $\mapsto$ не належать алфавіту A). При цьому формула $\alpha \to \beta$ називається *звичайною формулою підстановки*, а формула $\alpha \mapsto \beta$ – *заключною формулою підстановки*. Сама підстановка (як дія) задається формулою підстановки і застосовується до слів ω в алфавіті A. Суть операції $\alpha \to \beta$ ($\alpha \mapsto \beta$) полягає в відшуканні в слові ω найлівішого входження підслова α і заміні його на слово β. При цьому інші частини слова ω залишаються незмінними. Якщо ліва частина формули підстановки входить у слово ω, то кажуть, що дана *формула є застосовною до слова ω*. В іншому випадку кажемо, що *формула підстановки є незастосовною до слова ω*, і в цьому випадку підстановка не виконується. Якщо права частина формули підстановки – порожнє слово (яке в НАМ не познається ніяким символом), то підстановка $\alpha \to$ зводиться до видалення підслова α в слові ω. Якщо в лівій частині формули підстановки є порожнє слово, то підстановка $\to \beta$ полягає в дописуванні зліва до слова ω слова β. Зауважимо, що згідно означення

формула з порожньою лівою частиною застосовна до будь-якого слова.

5.1. Приклад. Нехай для слів в алфавіті $A = \{a, b, c\}$ задані наступні підстановки:

а) $ab \to ac$;
б) $abc \to ba$;
в) $ba \to$;
г) $b \mapsto c$.

Застосуємо кожну з них до слова *abbacba*.

Для застосування марківської підстановки до слова ω необхідно знайти найлівіше входження в ω лівої частини підстановки. У випадку а) слово *ab* тільки раз входить у слово *abbacba*, тому в результаті підстановки одержуємо слово *acbacba*. Оскільки слово *abc* не є підсловом слова *abbacba*, то підстановка з пункту б) є незастосовною до даного слова. У пунктах в) та г) ліві частини підстановок входять декілька раз у слово *abbacba*, а тому застосовуємо підстановки до першого входження їх у вхідне слово. Отримаємо слова *abcba* і *acbacba* відповідно.

5.2. Приклад. Застосувати кожну з підстановок із попереднього прикладу до слова *abbacba* максимальну кількість разів.

Застосувавши один раз підстановку $ab \to ac$ до слова *abbacba*, одержимо слово *acbacba*, яке вже не містить підслова *ab*, а тому дана підстановка далі є незастосовною. Підстановка з пункту б) є незастосовною до слова *abbacba*. Після першого застосування підстановки $ba \to$ до слова *abbacba* отримуємо слово *abcba*, до якого ще раз можна застосувати дану підстановку. Остаточно одержуємо слово *abc*, до якого вже незастосовна підстановка з пункту в). Оскільки підстановка $b \mapsto c$ є заключною, то вона застосовується тільки раз до вхідного слова і її результатом є слово *acbacba*.

Нормальним алгоритом Маркова називається пара (A, P), де A – алфавіт, а P – скінченна впорядкована послідовність формул

підстановки:

$$\begin{cases} \alpha_1 \to \beta_1 \\ \alpha_2 \to \beta_2 \\ \dots \\ \alpha_k \to \beta_k \end{cases},$$

яка називається *схемою підстановок*. При цьому вважається, що в P виділено деяку підмножину $P_f \subset P$ заключних формул підстановки.

Роботу НАМ можна описати наступним чином. Нехай задано деяке вхідне слово ω (у НАМ не важливо, де саме воно записано). Далі всі формули схеми підстановок проглядаються зверху вниз, обирається перша з формул, яка застосовна до слова ω, і виконується підстановка відповідно до знайденої формули підстановки. В результаті одержується нове слово ν. Потім ті ж самі дії виконуються з словом ν і т.д. При цьому зауважимо, що на кожному кроці формули в P завжди проглядаються зверху вниз, починаючи з першої формули! Якщо на черговому кроці була застосована звичайна формула підстановки, то робота НАМ продовжується, якщо ж – заключна формула, то після її застосування робота НАМ зупиняється і одержане слово називається *вихідним словом* або *результатом застосування НАМ до вхідного слова* ω. У випадку, якщо на деякому кроці кожна з формул схеми P є незастосовною до проміжного слова, то робота НАМ також зупиняється і вихідним словом вважається дане проміжне слово. В обох розглянутих випадках кажуть, що *НАМ застосовний до вхідного слова* ω.

5.3. Приклад. Застосувати кожен з НАМ

а) $\begin{cases} ab \to ba \\ ba \to \end{cases}$ б) $\begin{cases} ba \to \\ ab \mapsto ba \end{cases}$

в алфавіті $A = \{a, b\}$ до слова $abbbbaa$.

а) Спершу застосовуємо першу підстановку максимальну можливу кількість разів: $\underline{ab}bbbaa \Rightarrow b\underline{ab}bbaa \Rightarrow bb\underline{ab}baa \Rightarrow bbb\underline{ab}aa \Rightarrow bbbbaaa$. Потім застосуємо до слова $bbbbaaa$ підстановку $ba \to$: $bbbbaaa \Rightarrow bbbaa \Rightarrow bba \Rightarrow b$. В результаті одержуємо слово b.

Оскільки кожна з формул схеми а) є незастосовною до слова b, то НАМ застосовний до слова $abbbbaa$.

б) Застосуємо НАМ до слова $abbbbaa$: $abbb\underline{ba}a \Rightarrow abb\underline{ba} \Rightarrow \underline{ab}b \Rightarrow bab$. Оскільки ми використали заключну підстановку $ab \mapsto ba$, то робота НАМ зупиняється і алгоритм з пункту б) застосовний до слова $abbbbaa$.

Проте може трапитися так, що НАМ ніколи не зупиниться, тобто на кожному кроці до проміжного слова застосовна звичайна формула. В цьому випадку кажуть, що *НАМ є незастосовним до вхідного слова ω* або *зациклюється на вхідному слові ω*. *Область застосування НАМ* – це множина всіх слів, до яких застосовний даний НАМ.

5.4. Приклад. Визначимо область застосування НАМ відносно алфавіту $A = \{a, b\}$:

$$\begin{cases} a \to a \\ b \mapsto a \end{cases}$$

Перша формула застосовна до будь-якого вхідного слова, яке міститить хоча б одну букву a, причому вона не змінює даного слова. Тому на таких словах НАМ зациклюється. Якщо ж у вхідному слові немає букв a, але є хоча б одна буква b, тоді перша формула не буде застосовуватися, а відразу виконується друга підстановка і НАМ зупинить свою роботу. До порожнього вхідного слова кожна формула підстановоки є незастосовною, а тому НАМ до нього застосовний. Отже, область застосування даного НАМ – всі слова b^n $(n \geq 0)$.

5.5. Приклад. Побудуємо НАМ, який застосовний до всіх слів в алфавіті $A = \{a, b\}$, крім слів aa та ab.

З умови задачі випливає, що НАМ має зациклюватися на словах aa та ab. Однак алгоритм

$$\begin{cases} aa \to aa \\ ab \to ab \end{cases}$$

не є розв'язком даної задачі, оскільки він зациклюється не тільки на даних словах, але й на будь-яких інших словах, які містять підслово aa або ab. Зазвичай задачі такого типу розв'язують наступним

чином: початок і кінець вхідного слова ω відмічають спеціальними мітками (наприклад, $*\omega+$), а потім використовують формули підстановки, в лівій частині яких вказуються потрібні слова і ці мітки. Для зупинки алгоритму на інших словах можна використати, наприклад, формулу $* \mapsto$. Таким чином, шуканий НАМ має вигляд:

$$\begin{cases} +a \to a+ \\ +b \to b+ \\ *aa+ \to *aa+ \\ *ab+ \to *ab+ \\ * \mapsto \\ \mapsto *+ \end{cases}$$

Два НАМ над одним і тим же алфавітом A називаються *рівносильними*, якщо їх області застосувань співпадають і на однакових вхідних словах вони видають однакові результати.

5.6. Приклад. Визначити, чи є рівносильними наступні пари НАМ відносно алфавіту $A = \{a, b\}$:

а) $P_1 : \begin{cases} bb \to ab \end{cases}$ $P_2 : \begin{cases} bb \to ba \end{cases}$

б) $P_3 : \begin{cases} a \to ab \\ b \to aab \end{cases}$ $P_4 : \begin{cases} a \to b \\ b \to ba \end{cases}$

в) $P_5 : \begin{cases} b \mapsto ba \end{cases}$ $P_6 : \begin{cases} +a \to a+ \\ +b \mapsto ba \\ \to + \end{cases}$

Розглянемо схеми P_1 і P_2 НАМ з пункту а). У них одна і та ж область застосування – множина всіх слів в алфавіті $A = \{a, b\}$. Однак друга умова рівносильності (однакові результати на однакових вхідних словах) не виконується. Дійсно,

$$P_1 : \ a\underline{bb}b \Rightarrow aa\underline{bb} \Rightarrow aaab, \quad P_2 : \ a\underline{bb}b \Rightarrow abab.$$

Отже, НАМ з пункту а) не рівносильні.

У НАМ P_3 і P_4 області застосування співпадають і містять тільки єдине слово – порожнє. На непорожніх словах дані алгоритми зациклюються, причому по-різному. Однак така різна поведінка при зациклюванні не відіграє жодної ролі в означенні рівносильності алгоритмів. Важливо тільки, щоб у випадку зупинки алгоритми видавали однакові вихідні слова. А для пари з пункту б) ця умова виконується: на єдиному слові (порожньому), до якого вони застосовні, НАМ видають одну і ту ж відповідь – порожнє слово. Таким чином, дані алгоритми рівносильні.

В алгоритмів з пункту в), які замінюють перше входження букви b на ba, не співпадають області застосування: P_5 застосовний до всіх слів в алфавіті A, а P_6 зациклюється на словах, які не містять букви b. Отже, алгоритми з пункту в) не є рівносильними.

Рекомендована література : [**1**, с. 45–53], [**5**, с. 362–365], [**10**, с. 354–356], [**14**, с. 138–146], [**19**, с. 18–21].

Питання та вправи до параграфа 5.

5.1. Дайте означення формули підстановки в алфавіті A. Яка формула підстановки називається заключною?

5.2. Які НАМ називаються рівносильними?

5.3. Нормальний алгоритм Маркова $(\{a, b\}, P)$ задається схемою

$$P: \begin{cases} ba \to a \\ bb \to b \\ ab \to \\ \mapsto b \end{cases}.$$

Застосуйте його до слів $bbab$, $aaaab$, $bbaabba$.

5.4. В чому полягає робота НАМ (A, P), де $A = \{a, b\}$, а схема P має вигляд:

$$P: \begin{cases} a \to \\ b \to \\ \mapsto baba \end{cases}$$

5.5. Визначити область застосування НАМ з алфавітом $A = \{0, 1, 2\}$, якщо:

а) $P_1 : \begin{cases} 1 \to \\ 2 \to 2 \\ 0 \to \end{cases}$ б) $P_2 : \begin{cases} 1 \to \\ 22 \to 2 \\ 000 \to 00 \end{cases}$

в) $P_3 : \begin{cases} 1 \to 0 \\ 2 \to 1 \\ 00 \to \\ 0 \to 0 \end{cases}$ г) $P_4 : \begin{cases} 2 \to 1 \\ 1 \mapsto 0 \\ 00 \to \\ 0 \to 0 \end{cases}$

5.6. Знайти область застосування та визначити, в чому полягає робота НАМ $(\{a, b\}, P)$, де

$$P : \begin{cases} a \to aa \\ bb \to \\ b \mapsto ab \end{cases}$$

5.7. Побудувати НАМ в алфавіті $A = \{0, 1\}$, який застосовний тільки до порожнього слова та слова 1000.

5.8. Зі схеми

$$P : \begin{cases} 1 \to 0 \\ 01 \to 111 \\ 0 \to 1 \end{cases}$$

викреслити рівно одну формулу підстановки, щоб одержати НАМ, який застосовний до всіх слів в алфавіті $A = \{0, 1\}$.

5.9. Для кожної пари НАМ визначити, чи рівносильні вони відносно алфавіту $\{0, 1\}$:

а) $P_1 : \{1 \to 1$ $P_2 : \begin{cases} +1 \to 11 \\ 1 \to +1 \end{cases}$

б) $P_3 : \begin{cases} 10 \to 01 \\ 01 \to \end{cases}$ $P_4 : \begin{cases} 01 \to \\ 10 \to \end{cases}$

в) $P_5:\begin{cases}+1 \to 1\star\\ +0 \to 0+\\ \star 1 \mapsto\\ \star 0 \to 0+\\ \star \mapsto\\ \to +\end{cases}$ $P_6:\begin{cases}+0 \to 0+\\ +1 \mapsto\\ + \mapsto\\ 1 \to 1+\end{cases}$

5.10. З НАМ в алфавіті $\{0,1\}$, заданого схемою

$$P:\begin{cases}010 \to 001\\ 10 \to 01\\ 0101 \to 0011\\ 01 \to\end{cases}$$

викреслити якомога більше формул підстановки так, щоб одержаний алгоритм був рівносильний даному.

5.11. Побудувати машину Тюрінґа, яка рівносильна вказаному НАМ в алфавіті $\{0,1\}$:

а) $P:\begin{cases}+1 \to 0+\\ +0 \mapsto 0\\ \to +\end{cases}$ б) $P:\begin{cases}01 \mapsto 01\\ 10 \mapsto 10\\ \to\end{cases}$

5.12. Проаналізувати роботу нормального алгоритму Маркова, заданого над алфавітом $A=\{a_1,a_2,\ldots,a_m\}$ зі схемою

$$P:\begin{cases}** \to \#\\ \#a \to a\#, \text{ де } a\in A\\ \#* \to \#\\ \# \mapsto\\ *ab \to b*a, \text{ де } a,b\in A\\ \to *\end{cases}$$

§ 6. Синтез нормальних алгоритмів Маркова

У цьому параграфі на прикладах пояснюються основні прийоми побудови нормальних алгоритмів Маркова.

6.1. Приклад. Побудувати НАМ, який застосовний до всіх слів в алфавіті $A = \{a, b, c\}$ і видаляє із вхідного слова перше входження букви a (якщо таке є), а букви b замінює на c.

Перш за все зауважимо, що в НАМ, на відміну від машини Тюрінґа, легко реалізуються вставки, заміни та видалення букв. Наприклад, заміна букви b на букву c здійснюється за допомогою формули підстановки $b \to c$, а видалення букви a реалізується формулою $a \to$. При цьому проміжне слово розтягується або стискається автоматично. Шуканий алгоритм спершу має всі букви b замінювати на c, тому першою формулою має бути $b \to c$. Якщо букв b у слові вже не залишиться, то потрібно видалити перше входження букви a, використавши заключну підстановку $a \mapsto$. Таким чином, НАМ матиме вигляд:

$$\begin{cases} b \to c \\ a \mapsto \end{cases}$$

6.2. Приклад. Побудувати НАМ, який „сортує” непорожнє слово в алфавіті $A = \{0, 1\}$ (послідовність з цифр 0 і 1) в порядку незростання.

Дана задача розв'язується з допомогою НАМ, який містить єдину формулу підстановки $01 \to 10$. Дана формула міняє місцями 0 з кожною 1 доти, доки у слові праворуч від хоча б однієї цифри 0 є цифра 1. Формула стає незастосовною, коли праворуч від кожного 0 нема жодної 1, тобто коли вхідне слово відсортоване по незростанню. Наприклад,

$$\underline{01}10 \to 1\underline{01}0 \to 1100.$$

6.3. Приклад. Побудувати НАМ, який застосовний до всіх слів в алфавіті $A = \{a, b\}$ і видаляє останню букву непорожнього вхідного слова.

Для розв'язання задачі потрібно помітити останню букву, поставивши після неї новий спецзнак *. Але як помістити цей знак в

кінець слова? Робиться це наступним чином: спочатку дописуємо ліворуч до вхідного слова *, а потім „переганяємо" її в його кінець. Неважко помітити, що „перескакування" зірочки через букву – це заміна пари $*x$ на пару $x*$, яка здійснюється з допомогою формули підстановки $*x \to x*$. Після того як * опиниться вкінці слова, потрібно видалити останню букву і * та зупинити роботу алгоритму. Таким чином, одержуємо наступний НАМ:

$$\begin{cases} \to * \\ *a \to a* \\ *b \to b* \\ a* \mapsto \\ b* \mapsto \end{cases}$$

Проаналізуємо його роботу на слові abb:

$$abb \to *abb \to **abb \to ***abb \to ***abb \to \dots$$

Бачимо, що цей алгоритм постійно дописує зліва зірочки. Чому? Нагадаємо, що формула підстановки з порожньою лівою частиною застосовна завжди, тому перша формула буде застосовуватися нескінченно, блокуючи доступ до наступних формул. Звідси випливає дуже важливе правило: *якщо в НАМ є формула з порожньою лівою частиною, то вона має міститися вкінці НАМ*. З урахуванням цього алгоритм перепишеться так:

$$\begin{cases} *a \to a* \\ *b \to b* \\ a* \mapsto \\ b* \mapsto \\ \to * \end{cases}$$

Однак це ще не все: алгоритм зациклюється на порожньому вхідному слові, бо остання формула завжди буде застосовна. В чому причина помилки? Річ у тім, що ми ввели знак * для того, щоб помітити останню букву вхідного слова, а потім видалити * і цю букву. Щоб врахувати випадок порожнього слова, треба перед останньою формулою записати ще одну формулу, яка видаляє „одиноку" зірочку і зупиняє алгоритм. Таким чином, остаточно НАМ матиме вигляд:

$$\begin{cases} *a \to a* \\ *b \to b* \\ a* \mapsto \\ b* \mapsto \\ * \mapsto \\ \to * \end{cases}$$

6.4. Приклад. Побудувати НАМ над алфавітом $A = \{a, b\}$, який подвоює кожне входження букви b та дописує вкінці слова суфікс $baba$.

На перший погляд, щоб подвоїти кожну букву b, досить застосувати формулу $b \to bb$. Щоб переконатися, що це не так, розглянемо приклад:

$$\underline{b}ab \to \underline{b}bab \to \underline{b}bbab \to \dots$$

Помилка тут у тому, що після заміни першого входження букви b на bb ми не можемо відрізнити вже замінені букви b від тих, які ще не мінялися. Для вирішення цієї проблеми можна помітити зліва спецзнаком * ту букву b, яку потрібно в даний момент замінити, а після того як така заміна вже здійснена, спецзнак треба перемістити до наступної букви.

Таким чином, спершу слід розмістити ліворуч від вхідного слова спецзнак *, а потім „перескакувати" через кожну наступну букву, не змінюючи її, якщо ця буква a, або подвоюючи, якщо нею є b. Якщо праворуч від * вже не виявиться жодної букви, то спецзнак потрібно замінити на слово $baba$.

Отже, отримуємо наступний НАМ:

$$\begin{cases} *a \to a* \\ *b \to bb* \\ * \mapsto baba \\ \to * \end{cases}$$

Перевіримо його на вхідному слові bab:

$$bab \to *bab \to bb * ab \to bba * b \to bbabb* \to bbabbbaba.$$

6.5. Приклад. Побудувати НАМ над алфавітом $A = \{a, b\}$, який видаляє останнє входження букви b (якщо таке є).

Для того щоб помістити * поруч з останнім входженням букви b, можна спершу перемістити * в кінець слова, а потім перенести * справа наліво через букви a до найближчої букви b. При цьому потрібно врахувати, що вхідне слово може не містити букви b: якщо зірочка знову опиниться на початку слова, то її потрібно видалити і зупинитися. Реалізуємо дану ідею з допомогою наступного НАМ:

$$\begin{cases} *a \to a* \\ *b \to b* \\ a* \to *a \\ b* \mapsto \\ * \mapsto \\ \to * \end{cases}$$

Перевіримо роботу даного алгоритму на вхідному слові aba:

$$aba \to *aba \Rightarrow ab\underline{a*} \to ab\underline{*a} \to ab\underline{a*} \to \dots$$

Як бачимо, замість того, щоб рухатися справа наліво до найближчої букви b, зірочка почала „кружляти" навколо останньої букви слова. Чому? Річ у тім, що перші дві формули, які переміщують * праворуч, заважають третій формулі. Зауважимо, що перестановка цих формул не приведе до потрібного результату, бо тоді * почне „бігати" навколо першої букви вхідного слова. Помилка полягає в тому, що ми використовуємо спецзнак * як для руху ліворуч, так і для руху праворуч. Щоб виправити цю помилку, потрібно просто ввести ще один спецзнак, наприклад #, поділивши між цими спецзнаками обов'язки: нехай * переміщується праворуч, а # – ліворуч. З'явитися ж спецзнак # має тоді, коли * дійде до кінця слова, тобто коли справа від * не виявиться інших букв. Остаточно отримаємо такий НАМ:

$$\begin{cases} *a \to a* \\ *b \to b* \\ * \to \# \\ a\# \to \#a \\ b\# \mapsto \\ \# \mapsto \\ \to * \end{cases}$$

6.6. Приклад. Побудувати НАМ, який застосовний до всіх слів в алфавіті $A = \{a, b, c\}$ і визначає кількість різних букв у вхідному слові. Відповідь одержати в унарній системі числення (наприклад: $aabbab \to ||$).

Для розв'язання цієї задачі спочатку відсортуємо вхідне слово таким чином, щоб спершу були всі букви a, потім – b, а вкінці – c. Далі видалимо зайві букви так, щоб залишилась максимальна кількість попарно різних букв. Для цього скористаємось формулами підстановки виду $xx \to x$. Насамкінець замінимо всі букви на палички. Шуканий НАМ матиме вигляд:

$$\begin{cases} ba \to ab \\ ca \to ac \\ cb \to bc \\ aa \to a \\ bb \to b \\ cc \to c \\ a \to | \\ b \to | \\ c \to | \end{cases}$$

Наприклад,

$$aab\underline{ba}b \to aa\underline{ba}bb \to aaabbb \to aabbb \to abbb \to abb \to ab \to |b \to ||.$$

6.7. Приклад. Побудувати НАМ над алфавітом $A = \{0, 1, 2\}$, який дописує праворуч до непорожнього вхідного слова (трійкового числа) знак „= ” і стільки паличок, скільки цифр містить трійкове число.

Розв'яжемо дану задачу наступними чином. Спершу за кожною цифрою вхідного слова вставимо паличку $|$. Для цього допишемо зліва до вхідного слова спецзнак $*$, а потім перенесемо його через кожну цифру так, щоб зліва від нього залишалася ця цифра і відповідна їй паличка. Наприклад,

$$10 \to *10 \to 1|*0 \to 1|0|*$$

В одержаному слові переставимо цифри і палички так, щоб зліва опинилися всі букви, а справа – палички, зберігаючи при цьому вихідний взаємний порядок як букв, так і паличок:

$$1|0|* \to 10||*$$

Вкінці слова замінимо $*$ на новий спецзнак $=$ і перемістимо його ліворуч до першої цифри:

$$10||* \to 10|| = \to 10| = | \to 10 = ||$$

Всі вказані дії описуються наступним алгоритмом Маркова:

$$\begin{cases} *n \to n|*, \ n \in A \\ |n \to n|, \ n \in A \\ * \to = \\ | = \to = | \\ n = \mapsto n =, \ n \in A \\ \to * \end{cases}$$

Рекомендована література : [**10**, с. 354–356], [**14**, с. 146–180], [**19**, с. 21–32], [**21**, с. 178–189].

Питання та вправи до параграфа 6.

У задачах 6.1-6.21 побудувати НАМ, який застосовний до всіх слів в алфавіті A і виконує наступну роботу:

6.1. $A = \{$Н, П, У$\}$. Дописує справа до вхідного слова ω слово ПНУ ($\omega \to \omega$ПНУ).

6.2. $A = \{a, b\}$. Залишає у слові тільки першу букву (порожнє слово не міняє).

6.3. $A = \{a, b\}$. Залишає у слові тільки останню букву (порожнє слово не міняє).

6.4. $A = \{a, b, c\}$. За першою буквою непорожнього вхідного слова вставляє букву c.

6.5. $A = \{a, b\}$. Якщо у вхідному слові є принаймні дві букви, то міняє місцями перші дві букви.

6.6. $A = \{a, b\}$. Якщо вхідне слово містить більше букв a, ніж букв b, то видає вихідне слово з однієї букви a; якщо однакова кількість букв a і b, то видає порожнє слово; інакше – слово b.

6.7. $A = \{a, b\}$. У вхідному слові всі букви a замінює на b, а всі (попередні) букви b – на a.

6.8. $A = \{a, b, c\}$. Подвоює кожну букву у вхідному слові.

6.9. $A = \{a, b\}$. Дописує праворуч до вхідного слова стільки паличок, зі скількох підряд записаних букв b починається це слово.

6.10. $A = \{a, b\}$. Зі всіх входжень букви a у вхідне слово залишає тільки останнє входження, якщо таке є.

6.11. $A = \{a, b\}$. Якщо вхідне слово містить одночасно букви a і b, то замінює його на порожнє слово.

6.12. $A = \{a, b\}$. Якщо вхідне слово не є словом $aabba$, то замінює його на порожнє слово.

6.13. $A = \{a, b\}$. Визначає, чи входить перша буква непорожнього вхідного слова ще раз у це слово. Відповідь: слово a, якщо входить, або порожнє слово в протилежному випадку.

6.14. $A = \{a, b\}$. Переносить останню букву непорожнього вхідного слова на його початок.

6.15. $A = \{a, b\}$. В непорожньому вхідному слові переставляє місцями першу та останню букви.

6.16. $A = \{a, b\}$. Якщо в непорожньому вхідному слові співпадають перша та остання букви, то видаляє ці букви, в іншому випадку слово не змінює.

6.17. $A = \{a, b, c\}$. Якщо вхідне слово не містить букви c, то замінює всі букви a на b. В іншому випадку видає слово з однієї букви c.

6.18. $A = \{a, b\}$. Подвоює вхідне слово (дописує справа його копію).

6.19. $A = \{a, b\}$. Обертає вхідне слово (наприклад, $aab \to baa$).

6.20. $A = \{a, b\}$. Визначає, чи є слово поліндромом. Відповідь: слово a, якщо є, або порожнє слово в протилежному випадку.

6.21. $A = \{a, b\}$. Нехай вхідне слово має непарну довжину. Видаляє з нього середню букву.

§ 7. Нормально обчислювані функції

В даному параграфі розглядаються функції, для яких існує нормальний алгоритм Маркова, що їх обчислює. Зауважимо, що при аналізі деяких вхідних слів алгоритм Маркова може ніколи не зупинитися (зациклюється), і тому функція може бути не визначена для деяких вхідних слів-аргументів. У загальному випадку кажемо, що функція $f : (X^*)^n \to Y^*$ є *частково визначеною функцією*, якщо вона не визначена для деяких наборів аргументів $(x_1, \ldots, x_n) \in (X^*)^n$, тобто коли область визначення f є підмножиною $(X^*)^n$. Якщо область визначення f співпадає з $(X^*)^n$, то кажемо, що *функція f всюди визначена на $(X^*)^n$*. Якщо функція f не визначена на наборі $(x_1, \ldots, x_n)$, то писатимемо $f(x_1, \ldots, x_n) = \uparrow$.

Частково визначена функція $f : (X^*)^n \to Y^*$ називається *нормально обчислюваною*, якщо існує такий нормальний алгоритм Маркова, що для кожного набору $(x_1, \ldots, x_n) \in (X^*)^n$, на якому f визначена, НАМ перетворює вхідне слово $x_1 * x_2 * \ldots * x_n$ у вихідне слово y, де $y = f(x_1, \ldots, x_n) \in Y^*$, а $*$ – спеціальний допоміжний символ, який розділяє вхідні аргументи; і на кожному наборі $(x_1, \ldots, x_n) \in (X^*)^n$, для якого функція не визначена, НАМ зациклюється.

Надалі найчастіше розглядатимемо числові функції $f : (\mathbb{N}_0)^n \to \mathbb{N}_0$, де $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.

7.1. Приклад. Покажемо, що функція $f : \mathbb{N}_0 \to \mathbb{N}_0$, $f(n) = 0$ обчислюється з допомогою деякого НАМ.

Вважатимемо, що числа записані в двійковій системі числення. Для побудови НАМ, який обчислює дану функцію, достатньо в його схему підстановок включити формули для видалення всіх букв (цифр 0 і 1) вхідного слова і заключну формулу для заміни порожнього слова нулем. Таким чином, схема НАМ матиме вигляд:

$$\begin{cases} 0 \to \\ 1 \to \\ \mapsto 0 \end{cases}$$

7.2. Приклад. Покажемо, що функція $f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$, $f(n, m) = |n - m|$ є нормально обчислюваною (вважаємо, що невід'ємні цілі числа записані в унарній системі числення).

Нагадаємо, що

$$|n - m| = \begin{cases} n - m, \ n \geq m \\ m - n, \ n < m \end{cases}$$

На початку роботи вхідне слово має вигляд $\underbrace{||\ldots|}_{n} * \underbrace{||\ldots|}_{m}$. Оскільки $|n - m| = |(n - 1) - (m - 1)|$, то для розв'язання задачі досить одночасно видаляти по одній паличці зліва і справа доти, доки в деякій частині не залишиться жодної палички. Отже, НАМ задається схемою:

$$\begin{cases} |*| \to * \\ * \mapsto \end{cases}$$

Наприклад, $||*|||| \to |*||| \to *|| \to ||$ і $f(2, 4) = |2 - 4| = 2$.

7.3. Приклад. Покажемо, що функція $f : \mathbb{N}_0 \to \mathbb{N}_0$, $f(n) = n \bmod 4$, яка обчислює остачу від ділення натурального числа на 4, є нормально обчислюваною (вважаємо, що невід'ємні цілі числа записані в унарній системі числення).

Оскільки остачі від ділення чисел n і $n - 4$ на 4 однакові, то для знаходження остачі досить від числа віднімати 4 доти, доки

не отримаємо невід'ємне число менше 4 – воно і буде шуканою остачею. Таким чином, схема НАМ буде містити єдину формулу підстановки:

$$\Big\{|||| \to$$

Наприклад, $||||||||| \to ||||| \to |$ і $f(9) = 1$.

7.4. Приклад. Покажемо, що функція $f : \mathbb{N}_0 \to \mathbb{N}_0$, $f(n) = [n/4]$, є нормально обчислюваною.

Для знаходження цілої частини від ділення числа n на 4, досить спершу підрахувати, скільки четвірок паличок містить це унарне число, а потім витерти палички, які відповідають остачі. Дані міркування реалізуються такою схемою НАМ:

$$\begin{cases} *|||| \to |* \\ *| \to * \\ * \mapsto \\ \to * \end{cases}$$

Наприклад, $f(9) = 1$ і $||||||||| \to *|||||||| \to |*||||| \to ||*| \to ||* \to ||$.

7.5. Приклад. Комбінуючи попередні два приклади, покажемо, що функція $f : \mathbb{N}_0 \to \mathbb{N}_0 \times \mathbb{N}_0$, $f(n) = ([n/4],\ n \mod 4)$, є нормально обчислюваною.

Легко бачити, що дана функція обчислюється НАМ з схемою:

$$\begin{cases} *|||| \to |* \\ * \mapsto * \\ \to * \end{cases}$$

7.6. Приклад. Побудуємо НАМ, який обчислює функцію $f : \mathbb{N} \to \mathbb{N}$, $f(n) = n + 2$ (вважаємо, що натуральні числа задані в трійковій системі числення).

Для розв'язання задачі спершу введемо спецзнак * і перемістимо його вкінець слова, щоб помітити його останню букву (трійкову цифру). Далі додамо 2 за правилами трійкової арифметики. Якщо останньою цифрою є 0, то замінимо її на 2 і зупинимо роботу, інакше міняємо 1 на 0, 2 на 1 і переміщуємося ліворуч для додавання 1 до попередньої цифри з допомогою нового спецзнаку # (при цьому

враховуємо, що цією цифрою може бути 2, яку треба замінити на 0 і перемістити ліворуч для додавання 1).

Всі розглянуті дії описуються таким НАМ:

$$\begin{cases} *0 \to 0* \\ *1 \to 1* \\ *2 \to 2* \\ 0* \mapsto 2 \\ 1* \to \#0 \\ 2* \to \#1 \\ 0\# \mapsto 1 \\ 1\# \mapsto 2 \\ 2\# \to \#0 \\ \# \mapsto 1 \\ \to * \end{cases}$$

7.7. Приклад. Нехай вхідний алфавіт $A = \{0, 1, 2, 3\}$. Вважаючи вхідне слово записом числа в четвірковій системі числення, перевести це число в двійкову систему числення.

Як відомо, для переведення числа з четвіркової системи числення в двійкову, потрібно кожну четвіркову цифру замінити на пару відповідних їй двійкових цифр: $0 \to 00$, $1 \to 01$, $2 \to 10$, $3 \to 11$, див. § 2 першого розділу. Користуючись цим фактом, схему НАМ запишемо так:

$$\begin{cases} *0 \to 00* \\ *1 \to 01* \\ *2 \to 10* \\ *3 \mapsto 11* \\ * \mapsto \\ \to * \end{cases}$$

Нехай задано два нормальні алгоритми Маркова НАМ_1 і НАМ_2 в одному і тому ж алфавіті A, які обчислюють словесні функції $f_1 : A^* \to A^*$ та $f_2 : A^* \to A^*$ відповідно. Нас цікавитимуть НАМ,

які обчислюють їх композицію $f_2 \circ f_1$. Нагадаємо, що композицією функцій f_1 та f_2 називають таку функцію $f = f_2 \circ f_1$, що для кожного x з області визначення функції f_1 значення $f(x) = f_2(f_1(x))$. При цьому вимагається, щоб функція f_2 була визначена на значенні $f_1(x)$. Вихідне слово НАМ_1 можна подати на вхід НАМ_2. Таке послідовне виконання алгоритмів називається композицією НАМ_1 та НАМ_2 і позначається $\text{НАМ}_2(\text{НАМ}_1)$. Якщо будь-який з алгоритмів зациклюється, то зациклюватися має і їх композиція.

Доведена наступна теорема: *для будь-яких двох нормальних алгоритмів в алфавіті A, які обчислюють функції $f_1 : A^* \to A^*$ та $f_2 : A^* \to A^*$ відповідно, існує НАМ, який обчислює композицію $f_2 \circ f_1$*, див. [**14**]

7.8. Приклад. Побудуємо нормальний алгоритм, який є композицією наступних двох алгоритмів НАМ_1 і НАМ_2 з схемами P_1 і P_2 відповідно відносно алфавіту $\{a, b\}$:

$$P_1 : \begin{cases} *a \to a* \\ *b \to ab* \\ * \mapsto \\ \to * \end{cases} \qquad P_2 : \begin{cases} b \to \end{cases}$$

Спершу зауважимо, що в загальному випадку не можна будувати композицію простим виписуванням одна за одною формул підстановки з НАМ_1 і НАМ_2. Наприклад, якщо спочатку виписати формули з НАМ_1, а потім з НАМ_2, то отримаємо алгоритм НАМ_{12}, а якщо спершу виписати формули з НАМ_2, а потім з НАМ_1, то отримаємо алгоритм НАМ_{21}, які задані схемами:

$$P_{12} : \begin{cases} *a \to a* \\ *b \to ab* \\ * \mapsto \\ \to * \\ b \to \end{cases} \qquad P_{21} : \begin{cases} b \to \\ *a \to a* \\ *b \to ab* \\ * \mapsto \\ \to * \end{cases}$$

Але ні НАМ_{12}, ні НАМ_{21} не є композицією алгоритмів НАМ_1 і НАМ_2. Наприклад, для вхідного слова aaa маємо:

$$\text{НАМ}_{12}: \ aba \to *aba \to a*ba \to aab*a \to aaba* \to aaba$$

$$\text{НАМ}_{21}: \ aba \to aa \to *aa \to a*a \to aa* \to aa,$$

тоді як $\text{НАМ}_2(\text{НАМ}_1(aba))=\text{НАМ}_2(aaba) = aaa$.

Зауважимо, що існує загальний метод побудови композиції НАМ, див. [**14**, с. 198]. Однак цей спосіб досить громіздкий і для простіших НАМ краще використовувати інший підхід: перш за все треба зрозуміти, яку задачу розв'язує кожен з алгоритмів НАМ_1 і НАМ_2, а потім послідовність цих задач об'єднати в спільну задачу і побудувати алгоритм для цієї загальної задачі.

В нашому прикладі алгоритм НАМ_1 замінює кожну букву b словом ab, а потім алгоритм НАМ_2 видаляє букви b. Зрозуміло, що послідовне застосування спершу НАМ_1, а потім НАМ_2 замінює кожну букву b вхідного слова буквою a. Шукана композиція НАМ_1 і НАМ_2 задається простою схемою НАМ:

$$\begin{cases} b \to a \end{cases}$$

Ми показали, що клас нормально обчислюваних функцій є досить широким. Поняття нормально обчислюваної функції – одне з основних понять теорії алгоритмів. Зазначимо, що з одного боку, кожна нормально обчислювана функція є обчислюваною за певною процедурою, яка відповідає інтуїтивному уявленню алгоритму, а з іншого – які б досі не будувалися класи точно визначених алгоритмів, завжди з'ясовувалося, що числові функції, які обчислюються за алгоритмами цих класів, є нормально обчислюваними. Тому загальноприйнятою є така наукова гіпотеза:

Теза Маркова-Черча. *Будь-який алгоритм можна реалізувати з допомогою НАМ.*

У формулювання цієї тези входить інтуїтивне поняття алгоритму, а тому її не можна ні довести, ні спростувати в загальноматематичному значенні. Справедливість гіпотези підтверджується

практикою: всі відомі алгоритми, які були придумані впродовж багатьох тисячоліть історії математики, можуть бути задані з допомогою НАМ. Проте справедливою є наступна теорема, доведення якої читач може знайти в [**10**]:

7.9. Теорема. *Для функції f наступні умови рівносильні:*

- *функція f є обчислюваною за Тюрінґом;*
- *функція f частково рекурсивна;*
- *функція f є нормально обчислюваною.*

Рекомендована література : [**10**, с. 356–361], [**14**, с. 180–186, 198–219].

Питання та вправи до параграфа 7.

7.1. Що ви розумієте під композицією НАМ?

7.2. Побудувати НАМ, який обчислює числову функцію $f : \mathbb{N}_0 \to \mathbb{N}_0$, $f(n) = n \mod 2$, якщо аргументи задані:

а) в унарній системі числення;
б) в двійковій системі числення;
в) в трійковій системі числення.

7.3. Побудувати НАМ, який обчислює числову функцію $f : \mathbb{N}_0 \to \mathbb{N}_0$, аргументи якої задані в двійковій системі числення.

а) $f(n) = 4_{10} \cdot n$;
б) $f(n) = [\frac{n}{3}]$;
в) $f(n) = [\frac{n+3}{4}] + 2$.

7.4. Побудувати НАМ, який обчислює числову функцію $f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$, аргументи якої задані в унарній системі числення.

а) $f(n, m) = n + m$;
б) $f(n, m) = \max\{n, m\}$;
в) $f(n, m) = \min\{n, m\}$.

7.5. Показати, що наступні частково визначені числові функції $f : \mathbb{N}_0 \to \mathbb{N}_0$, аргументи яких задані в четвірковій системі числення, є нормально обчислювані:

а) $f(n) = n + 1$;
б) $f(n) = n + 2$;
в) $f(n) = n - 2$.

7.6. Показати, що наступний НАМ обчислює функцію $f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$, $f(n,m) =$ НСД(n,m), аргументи якої задані в унарній системі числення:

$$\begin{cases} |a \to a| \\ |*| \to a* \\ |* \to *b \\ b \to | \\ a \to c \\ c \to | \\ * \to \end{cases}$$

7.7. $A = \{0,1,2,a\}$. З'ясувати, чи є вхідне слово записом числа в трійковій системі числення. Відповідь: 1 – якщо є, 0 – в протилежному випадку.

7.8. $A = \{0,1,2\}$. Вважаючи вхідне слово записом числа в трійковій системі числення, видалити з нього всі незначущі нулі.

7.9. $A = \{0,1\}$. Вважаючи непорожнє слово записом двійкового числа, визначити, чи є воно степенем двійки. Відповідь: 1 – якщо є, 0 – в протилежному випадку.

7.10. $A = \{0,1\}$. Вважаючи непорожнє слово записом двійкового числа, перевести його в четвіркову систему числення. (Зауваження: врахувати, що в двійковому числі може бути непарна кількість цифр.)

7.11. $A = \{|\}$. Перевести число з унарної системи числення в трійкову. (Рекомендація: можна у циклі видаляти з унарного числа по паличці і кожен раз додавати 1 до трійкового числа, яке на початку покласти рівним 0.)

7.12. Для кожної пари НАМ побудувати їх композиції відносно алфавіту $\{0,1\}$:

а) $P_1 : \begin{cases} 01 \to 10 \end{cases}$ $\qquad$ $P_2 : \begin{cases} 1 \to 0 \end{cases}$

б) $P_3:\begin{cases}*0 \to *\\ *1 \to 1*\\ * \mapsto \\ \to *\end{cases}$ $P_4:\begin{cases}1 \to 0\end{cases}$

в) $P_5:\begin{cases}*0 \to 1*\\ *1 \to 0*\\ * \mapsto \\ \to *\end{cases}$ $P_6:\begin{cases}01 \to\end{cases}$

7.13. Чи є НАМ зі схемою P_3 композицією нормальних алгоритмів зі схемами P_1 і P_2 відносно алфавіту $\{0,1,2\}$?

$P_1:\begin{cases}01 \to 10\end{cases}$ $P_2:\begin{cases}02 \to 20\end{cases}$

$P_3:\begin{cases}01 \to 10\\ 02 \to 20\end{cases}$

§ 8. Складність алгоритмів

В загальній теорії алгоритмів вивчається лише теоретична можливість розв'язання задач. При розгляді конкретних задач не звертається увага на ресурси часу і пам'яті для відповідних алгоритмів-розв'язків. Але теоретична можливість розв'язання задачі не гарантує її практичної реалізації.

Введемо необхідні означення, відштовхуючись від машин Тюрінґа. Нехай машина Тюрінґа обчислює словесну функцію $f(\omega)$. Визначимо функцію $t_T(\omega)$, значення якої для слова ω дорівнює кількості тактів машини Тюрінґа T, які виконуються при обчисленні $f(\omega)$, якщо $f(\omega)$ визначене. Якщо $f(\omega)$ не визначене, то значення функції $t_T(\omega)$ теж вважається не визначеним. Функція $t_T(\omega)$ називається *часовою складністю машини Тюрінґа T*.

Активною зоною при роботі машини Тюрінґа на слові ω називається множина всіх комірок стрічки, які містять непорожні букви або відвідувались головкою машини Тюрінґа в процесі обчислення. Введемо функцію $s_T(\omega)$, значення якої дорівнює довжині активної

зони при роботі машини Тюрінґа T на слові ω, якщо $f(\omega)$ визначене. В іншому випадку значення функції $s_T(\omega)$ не визначене. Функція $s_T(\omega)$ називається *ємнісною складністю машини Тюрінґа T*.

Введені функції $t_T(\omega)$ і $s_T(\omega)$ є словесними. Зручно ввести для розгляду функції натурального аргументу, поклавши:

$$t_T(n) = \max_{|\omega|=n}\{t_T(\omega)\} \quad \text{і} \quad s_T(n) = \max_{|\omega|=n}\{s_T(\omega)\}.$$

Ці функції теж називаються функціями часової і ємнісної складності (в найгіршому випадку) машини Тюрінґа T.

Визначимо функції часової та ємнісної складності для алгоритмів Маркова. Нехай алгоритм Маркова M обчислює словесну функцію $f(\omega)$. Визначимо функцію $t_M(\omega)$, значення якої для вхідного слова ω дорівнює кількості підстановок, які виконуються в процесі обчислення вихідного слова $f(\omega)$ з вхідного слова ω, якщо алгоритм застосовний до вхідного слова ω. Якщо ж алгоритм зациклюється на вхідному слові ω, то значення функції $t_M(\omega)$ вважається не визначеним. Функція $t_M(\omega)$ називається *часовою складністю алгоритму Маркова M*.

Введемо функцію $s_M(\omega)$, значення якої дорівнює максимальній з довжин слів, які виникають при обчисленні з вхідного слова ω вихідного слова $f(\omega)$, якщо $f(\omega)$ визначене. Якщо $f(\omega)$ не визначене, то значення функції $s_M(\omega)$ теж не визначене. Функція $s_M(\omega)$ називається *ємнісною складністю алгоритму Маркова M*.

Аналогічно, як і для машини Тюрінґа, зручно ввести функції натурального аргументу, поклавши:

$$t_M(n) = \max_{|\omega|=n}\{t_M(\omega)\} \quad \text{і} \quad s_M(n) = \max_{|\omega|=n}\{s_M(\omega)\}.$$

Ці функції теж називаються функціями часової і ємнісної складності (в найгіршому випадку) алгоритму Маркова M.

Гранична поведінка функцій t_T і t_M (відповідно s_T і s_M) при збільшенні розміру n задачі називається *асимптотичною часовою* (відповідно *ємнісною*) *складністю*. Для конкретних задач розглядаються, як правило, асимптотичні функції складності.

Нехай f і g – дві функції натурального аргументу. Кажуть, що $f(n) = O(g(n))$ (читається: „еф від ен є о велике від же від ен"), якщо існує така константа $c > 0$, $c \in \mathbb{R}$ і таке число $n_0 \in \mathbb{N}_0$, що $0 \le f(n) \le cg(n)$ для всіх $n \ge n_0$. Запис $f(n) = \Theta(g(n))$ означає,

що існують такі константи $c_1, c_2 > 0$, $c_1, c_2 \in \mathbb{R}$ і таке $n_0 \in \mathbb{N}_0$, що $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ для всіх $n \geq n_0$.

Кажуть, що машина Тюрінґа (алгоритм Маркова) розв'язує задачу за *поліноміальний час*, якщо $t_T(n) = O(p(n))$ ($t_M(n) = O(p(n))$) для деякого многочлена p. В протилежному випадку кажуть, що машина Тюрінґа (алгоритм Маркова) розв'язує задачу за *експоненціальний час*.

Часова складність алгоритму відображає потрібні для його роботи витрати часу. Аналогічно, як для машини Тюрінґа і алгоритмів Маркова, для будь-яких алгоритмічних моделей часова складність визначається як функція, яка кожній послідовності вхідних даних довжини n ставить у відповідність максимальний (за всіма індивідуальними задачами довжиною n) час $t(n)$, який витрачає алгоритм для розв'язання індивідуальних задач із цією довжиною. Про конкретну задачу кажуть, що вона розв'язується за поліноміальний час, якщо існує алгоритм (наприклад, машина Тюрінґа чи алгоритм Маркова), який розв'язує цю задачу за поліноміальний час. Через $\mathbb{P}$ позначається клас всіх задач, які розв'язуються за поліноміальний час. В іншому випадку кажуть, що задача розв'язується за експоненціальний час. Задачу називають *важкорозв'язною*, якщо немає поліноміального алгоритму для її розв'язання.

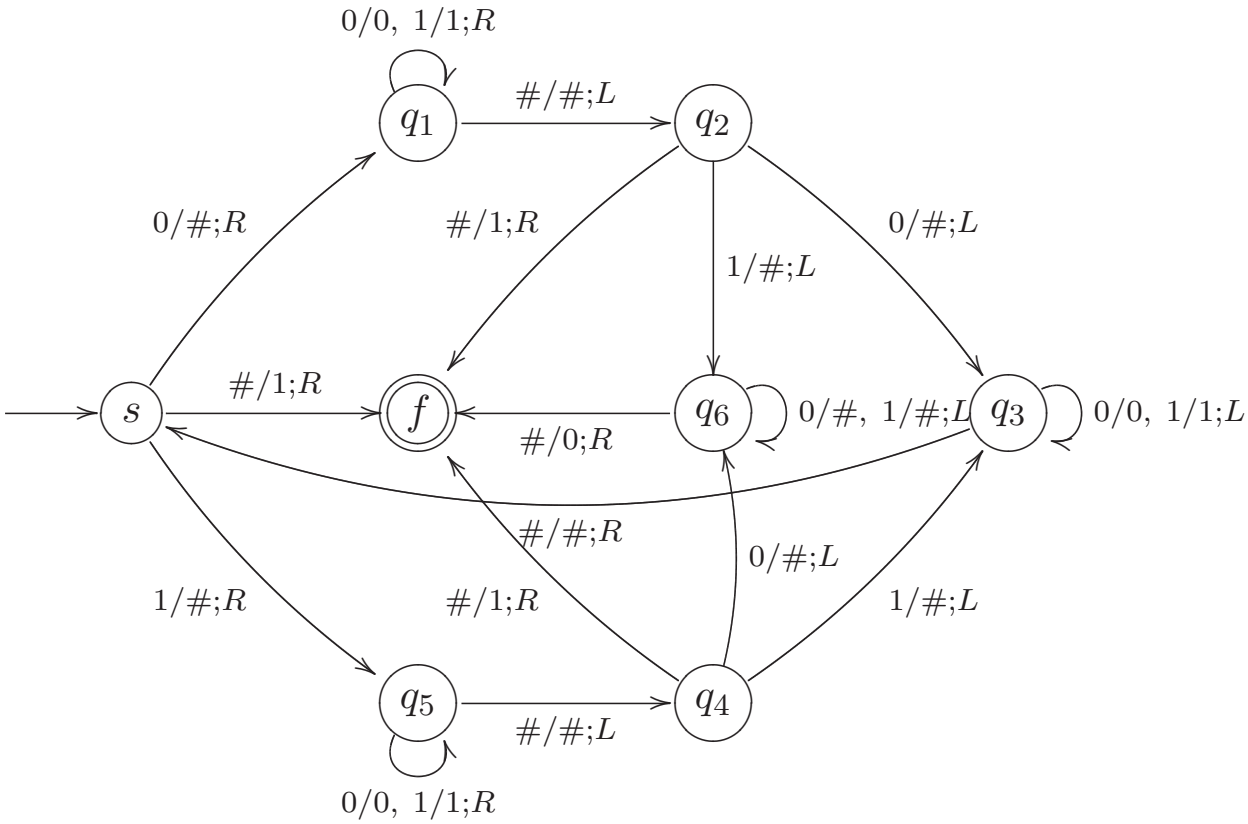
Вважається, що поліноміальні алгоритми відповідають швидким, ефективним на практиці алгоритмам, а поліноміально розв'язні задачі відповідають легким задачам, які можуть бути розв'язаними за прийнятний час на входах довжини, що має практичний інтерес. Часто поліноміальний алгоритм справді задовольняє всі практичні потреби. В гіршому ж випадку його можна вважати швидким лише асимптотично, а на відносно невеликих входах алгоритм може працювати довго. Слід підкреслити, що клас поліноміально розв'язних задач не залежить від того, яка саме з багатьох можливих формалізацій алгоритму вибрана. Цей факт можна строго довести як теорему для будь-яких означень алгоритму.

Слід зазначити, що є експоненціальні алгоритми, які добре зарекомендували себе на практиці. Річ у тім, що часову складність означено як міру поведінки алгоритму в найгіршому випадку. Насправді може виявитись, що для розв'язання більшості конкретних

задач потрібно значно менше часу, і це дійсно так для деяких добре відомих алгоритмів. Наприклад, симплекс-метод для розв'язування задач лінійного програмування має експоненціальну часову складність, але дуже добре працює на практиці.

8.1. Приклад. Побудувати машину Тюрінґа, яка в алфавіті $A = \{0, 1\}$ для будь-якого слова $P = p_1p_2 \dots p_n$ визначає, чи є воно симетричним, тобто $P = p_1p_2 \dots p_n = p_np_{n-1} \dots p_1 = P'$. Відповідь: слово 1, якщо слово P симетричне, 0 – в протилежному випадку. Оцінити часову та ємнісну складність побудованої машини Тюрінґа.

Граф шуканої машини Тюрінґа має вигляд:



Робота машини Тюрінґа здійснюється циклами. Впродовж першого циклу машина перевіряє, чи виконується рівність $p_1 = p_n$. Для цього в стані s запам'ятовується буква p_1 і головка машини Тюрінґа в станах q_1 або q_5 зміщується праворуч доти, доки не знаходить букву p_n і порівнює її з p_1 в станах q_2 або q_4. Якщо ці букви різні, то машина переходить в стан q_6, видаляє слово на стрічці і друкує букву 0, в протилежному випадку машина здійснює другий цикл, впродовж якого порівнює букви p_2 та p_{n-1} і т. д. Зрозуміло, що найбільшу кількість тактів машина Тюрінґа виконує на симетричних

словах довжини n. Якщо симетричне слово P має парну довжину, то буде проведено $n/2$ повних циклів порівнянь букв, впродовж яких буде здійснено число тактів, яке дорівнює

$$(2n+1)+(2(n-2)+1)+\ldots+5=\frac{(2n+1)+5}{2}\cdot\frac{n}{2}=\frac{(n+3)n}{2}.$$

Після цього машина здійснює ще один такт і зупиняє роботу.

Якщо ж симетричне слово P має непарну довжину, то буде проведено $(n-1)/2$ повних циклів порівнянь букв, впродовж яких буде здійснено число тактів, яке дорівнює

$$(2n+1)+(2(n-2)+1)+\ldots+7=\frac{(2n+1)+7}{2}\cdot\frac{n-1}{2}=\frac{(n+4)(n-1)}{2}.$$

Далі МТ здійснює ще три такти і переходить у кінцевий стан.

З проведених вище міркувань і аналізу графа машини Тюрінґа випливає, що $s_T(n)=n+1\le 2n$, $s_T(n)=O(n)$, а функція часової складності має вигляд

$$t_T(n)=\begin{cases}\frac{(n+3)n}{2}+1, n=2k, k\in\mathbb{N}\\ \frac{(n+4)(n-1)}{2}+3, n=2k-1, k\in\mathbb{N}\end{cases}=\frac{n^2+3n+2}{2}$$

Оскільки $\frac{n^2+3n+2}{2}\le\frac{n^2+3n^2+2n^2}{2}=3n^2$, то $t_T(n)=O(n^2)$ і задача розв'язується за поліноміальний час, тобто належить класу $\mathbb{P}$.

8.2. Приклад. Проаналізувати роботу алгоритму Маркова, заданого над алфавітом $A=\{a_1,a_2,\ldots,a_m\}$, та оцінити його складність:

$$\begin{cases}** \to \#\\ \#a\to a\#,\ \text{де } a\in A\\ \#*\to\#\\ \#\mapsto\\ *ab\to b*a,\ \text{де } a,b\in A\\ \to *\end{cases}$$

Якщо $P=b_1b_2\ldots b_n$, то оберненням слова P називається слово $P'=b_nb_{n-1}\ldots b_1$. Покажемо, що алгоритм здійснює обернення слів в алфавіті A. Дійсно, нехай маємо деяке слово P в алфавіті A. Тоді:

$P\to *P\to b_2*b_1\ldots b_n\to b_2b_3*b_1b_4\ldots b_n\to\ldots\to b_2b_3\ldots b_n*b_1\to *b_2b_3\ldots b_n*b_1\to\ldots\to b_3b_4\ldots b_n*b_2*b_1\to\ldots\to *b_n*b_{n-1}\ldots*$

$b_2*b_1 \to **b_n*b_{n-1}\ldots*b_2*b_1 \to \#b_n*b_{n-1}\ldots*b_2*b_1 \to b_n\#*b_{n-1}\ldots*$
$b_2*b_1 \to b_n\#b_{n-1}\ldots*b_2*b_1 \to \ldots \to b_nb_{n-1}\ldots b_1\# \mapsto b_nb_{n-1}\ldots b_1$

Оцінимо часову та ємнісну складність цього алгоритму. Оскільки кількість підстановок, які виконуються при обчисленні вихідного слова P' з вхідного слова P, залежить тільки від довжини слова, то функція часової складності $t_M(n) = t_M(\omega)$ для будь-якого слова ω довжини n. Знайдемо кількість підстановок при аналізі слова $P = b_1b_2\ldots b_n$ довжини n. На першому етапі алгоритм дописує зліва до слова $*$ і переносить її разом з першою буквою вкінець слова, використовуючи при цьому n підстановок. Далі алгоритм дописує зліва до одержаного слова $*$ і переносить її з другою буквою початкового слова ω до попередньої $*$, використовуючи $n-1$ підстановок і т.д., доки не отримаємо слово вигляду $*b_n * b_{n-1}\ldots * b_2 * b_1$. При цьому кількість підстановок дорівнює $n + (n-1) + \ldots + 1 = \frac{n(n+1)}{2}$. Далі зліва до слова дописується ще одна $*$ і $**$ міняється на $\#$, після чого решітка переноситься вкінець слова, знищуючи при цьому всі зірочки. Насамкінець виконується підстановка $\# \mapsto$ і алгоритм Маркова зупиняє свою роботу. Таким чином, на другому етапі виконується $2 + 2n$ підстановок. Отже,

$$t_M(n) = \frac{n(n+1)}{2} + 2 + 2n = \frac{n^2 + 5n + 4}{2} = O(n^2).$$

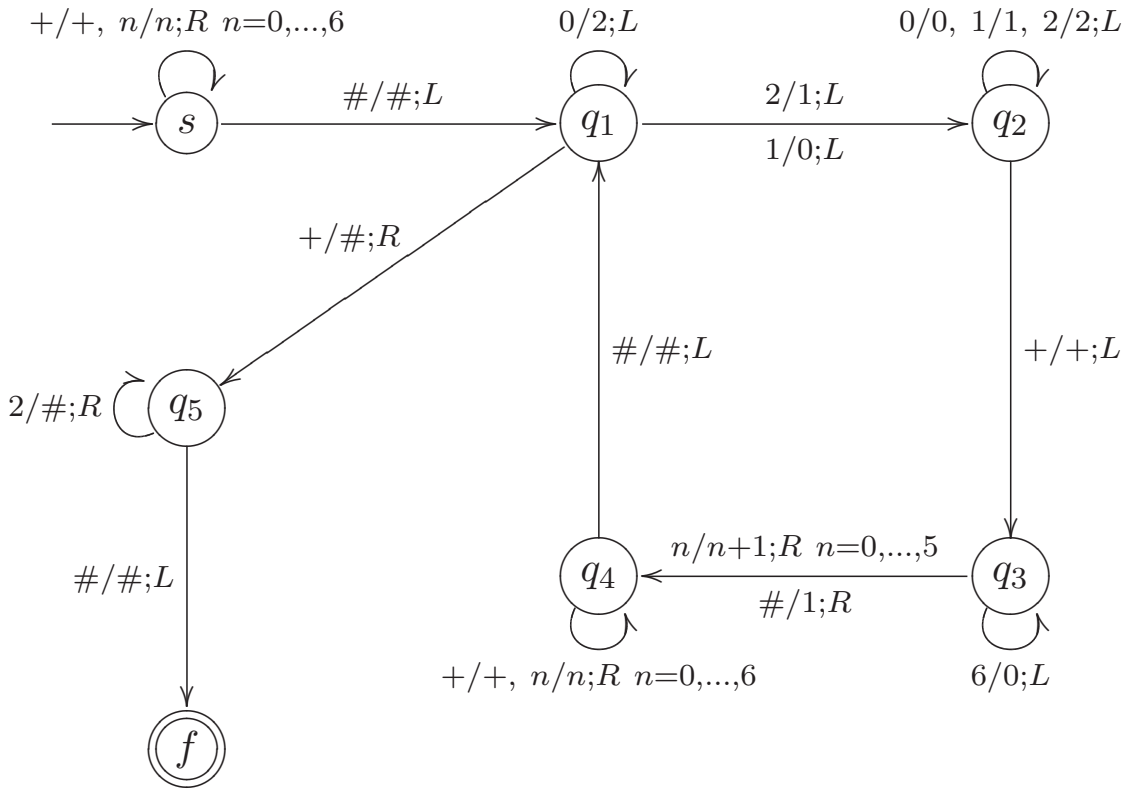
З виведення видно, що $**b_n * b_{n-1}\ldots * b_2 * b_1$ – слово максимальної довжини, тому $s_M(n) = 2n + 1 = O(n)$.

Таким чином, задача розв'язується за поліноміальний час і належить класу $\mathbb{P}$.

8.3. Приклад. Знайти довжину (кількість цифр) натурального числа, записаного в m-ковій системі числення, тобто знайти функцію, яка кожному числу $n = a_km^k + a_{k-1}m^{k-1} + \ldots a_1m + a_0$, де $a_k \neq 0$ і $0 \leq a_i < m$, ставить у відповідність число $|\overline{a_ka_{k-1}\ldots a_0}| = k + 1$.

Оскільки маємо, що $m^k \leq a_km^k + a_{k-1}m^{k-1} + \ldots + a_1m + a_0 \leq (m-1)m^k + (m-1)m^{k-1} + \ldots + (m-1)m + (m-1) = m^{k+1} - 1 < m^{k+1}$, то $k \leq \log_m n < k + 1$. Таким чином, $k = [\log_m n]$ і довжина натурального числа n, записаного в m-ковій системі числення, дорівнює $1 + [\log_m n] = 1 + [\frac{\ln n}{\ln m}] = O(\ln n)$. З останньої рівності видно, що асимптотична оцінка довжини числа не залежить від того, в якій системі числення воно записане.

8.4. Приклад. Побудувати машину Тюрінґа для визначення суми чисел, заданих у сімковій та трійковій системах числення (результат отримати в сімковій системі числення) і оцінити її складність.



Очевидно, що алфавіт $A = \{0, 1, 2, 3, 4, 5, 6, +\}$. Множина станів складається з семи елементів: $Q = \{s, q_1, q_2, q_3, q_4, q_5, f\}$. Нехай доданки відокремлюються знаком $+$, порожні секції, як і раніше, позначатимемо $\#$. На стрічці ліворуч від знака $+$ розміщено доданок у сімковій системі числення, а праворуч – доданок у трійковій системі числення. Ідея розв'язання задачі така: спочатку в стані s переміщуємося до останнього символу, потім від розміщеного праворуч трійкового числа в стані q_1 віднімаємо одиницю, діючи за правилами трійкової арифметики, після чого в стані q_2 переміщуємося ліворуч та до сімкового числа додаємо одиницю в стані q_3, діючи за правилами сімкової арифметики. Далі повертаємося до правого числа з допомогою стану q_4 і переходимо в стан q_1. Повторюємо всі операції доти, доки розміщений праворуч від знака $+$ доданок не буде вичерпано.

Оцінимо складність даного алгоритму. Очевидно, що функція $t_T(\omega)$ буде приймати максимальне значення на словах ω довжини n виду: $a + b_1 b_2 \dots b_{n-2}$. Оцінимо, на стільки може збільшитися

довжина слова в процесі додавання. Оскільки

$$\overline{b_1 b_2 \dots b_{n-2}} = b_1 3^{n-3} + b_2 3^{n-4} + \dots + b_{n-3} 3 + b_{n-2} < 3^{n-2},$$

то при додаванні цього числа в ліву частину загальна довжина слова збільшиться не більше, ніж на $\log_7 3^{n-2} = (n-2)\log_7 3$. Враховуючи крайні порожні клітинки і той факт, що кількість циклів, які здійснить машина Тюрінґа при додаванні, не перевищує 3^{n-2}, маємо, що ємнісна складність

$$s_T(n) \le n \log_7 3 - \log_7 9 + n + 2 = O(n),$$

а часова складність

$$t_T(n) \le 2(n \log_7 3 - \log_7 9 + n + 2)3^{n-2} = O(n3^n).$$

Отже, дана машина Тюрінґа розв'язує задачу за експоненціальний час.

Рекомендована література : [**5**, с. 374–384], [**12**, с. 129–204], [**16**, с. 330–353].

Питання та вправи до параграфа 8.

8.1. Дайте означення (асимптотичної) часової і ємнісної складності машини Тюрінґа та алгоритму Маркова.

8.2. Коли кажуть, що певна задача розв'язується за поліноміальний час?

8.3. Яка задача називається важкорозв'язною?

8.4. Чи можна стверджувати, що а) $2^{n+1} = O(2^n)$; б) $2^{2n} = O(2^n)$?

8.5. Довести, що для полінома $f(n) = a_m n^m + \dots + a_1 n + a_0$, де $a_m > 0$, $a_i \in \mathbb{R}$, $i = \overline{1,m}$, має місце рівність $f(n) = \Theta(n^m)$.

8.6. Доведіть, що $f(n) = n^{O(1)}$ тоді і тільки тоді, коли існує таке додатнє m, для якого $f(n) = O(n^m)$ (вважаємо, що $f(n) > 1$).

8.7. Порівняти час сортування масиву з 1 млн. чисел на двох комп'ютерах, перший з яких виконує 100 млн. операцій за секунду, а другий – 1 млн. Для першого комп'ютера складений алгоритм з складністю $2n^2$ операцій, а для другого – $50nlog_2n$.

8.8. При якому найменшому значенні n алгоритм, що вимагає $100n^2$ операцій, ефективніший за алгоритм, що вимагає 2^n операцій?

8.9. Дослідити, якою буде двійкова довжина числа, одержаного а) додаванням, б) множенням n додатних чисел, двійкова довжина кожного з яких не перевищує k.

8.10. Нехай A і B – $m_1 \times m_2$ і $m_2 \times m_3$ матриці відповідно. Визначити кількість арифметичних операцій, необхідних для обчислення їх добутку за звичайним алгоритмом. Оцінити складність.

8.11. Визначити найефективнішу послідовність множення матриць A, B, C, якщо їх розмірності відповідно

а) 10×5, 5×50, 50×1;

б) 20×50, 50×10, 10×40.

8.12. Визначити найефективнішу послідовність множення матриць A, B, C, D, якщо їх розмірності відповідно

а) 10×2, 2×5, 5×20, 20×3;

б) 20×5, 5×10, 10×40, 40×3.

8.13. Побудувати поліноміальний нормальний алгоритм Маркова в алфавіті $\{a, b\}$ з часовою складністю $O(1)$, який видаляє у вхідному слові, яке містить не менше ніж три букви, третю букву?

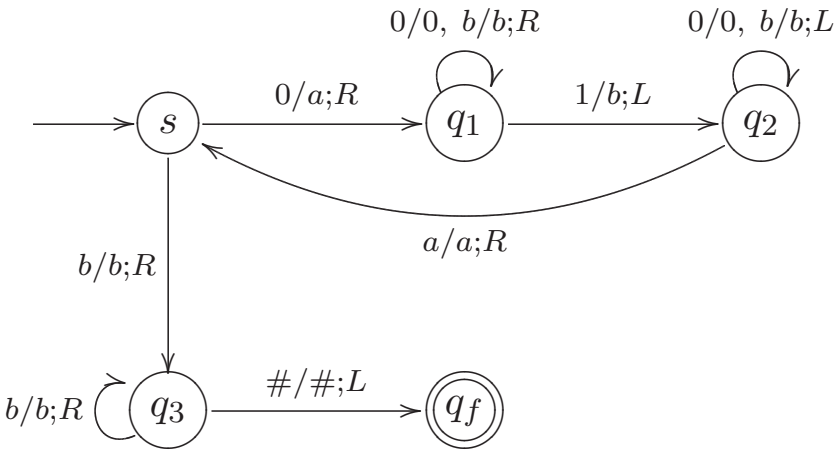
8.14. Побудувати ефективні нормальний алгоритм Маркова та машину Тюрінґа, які обчислюють функцію $f(n) = n - 3$ натуральних четвіркових аргументів. Оцінити їх часову та ємнісну складність.

8.15. Нормальний алгоритм Маркова (A, P) задається схемою

$$P: \begin{cases} ba\# \to a\#b, \text{ де } a, b \in A \\ *a \to a\#a*, \text{ де } a \in A \\ \# \to + \\ + \to \\ * \mapsto \\ \to * \end{cases}$$

Оцінити його часову та ємнісну складність.

8.16. Оцінити часову та ємнісну складність МТ, заданої графом:



8.17. Показати, що наступний НАМ обчислює функцію $f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$, $f(n, m) = nm$, аргументи якої задані в унарній системі числення:

$$\begin{cases} b| \to |b \\ a| \to |ba \\ a \to \\ |* \to *a \\ *| \to * \\ * \to \\ b \to | \end{cases}.$$

Оцінити його часову та ємнісну складність.

Список літератури

1. Алферова З.В. Теория алгоритмов / З.В. Алферова – М.: «Статистика», 1973. – 164 с.
2. Ахо А. Теория синтаксического анализа, перевода и компиляции / А. Ахо, Дж. Ульман. – М.: Мир, 1978. – Т. 1. – 611 с.
3. Безущак О.О. Елементи теорії чисел: Навчальний посібник / О.О. Безущак, О.Г. Ганюшкін. – К.: Видавничо-поліграфічний центр «Київський університет», 2003. – 202 с.
4. Белоусов А.И. Дискретная математика: Учеб. для вузов / А.И. Белоусов, С.Б. Ткачев. – 3-е изд. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2004. – 744 с.
5. Бондаренко М.Ф. Комп'ютерна дискретна математика: підручник / М.Ф. Бондаренко, Н.В. Білоус, А.Г. Руткас. – Харків: «Компанія СМІТ», 2004. – 480 с.
6. Гаврилків В.М. Регулярні вирази у програмних продуктах: навчальний посібник / В.М. Гаврилків. – Івано-Франківськ: «Сімик», 2012. – 72 с.
7. Гаврилов Г.П. Задачи и упражнения по дискретной математике: Учеб. пособие / Г.П. Гаврилов, А.А. Сапоженко. – 3-е изд., перераб. – М.: ФИЗМАТЛИТ, 2005. – 416 с.
8. Гашков С.Б. Современная элементарная алгебра в задачах и решениях / С.Б. Гашков. – М.: МЦНМО, 2006. – 328 с.
9. Завало С.Т. Алгебра і теорія чисел, ч. 2 / С.Т. Завало, В.М. Костарчук, Б.І. Хацет. – К.: Вища школа, 1976. – 384 с.
10. Игошин В.И. Математическая логика и теория алгоритмов: учеб. пособие / В.И. Игошин. – М.: Изд. центр «Академия», 2008. — 448 с.
11. Капітонова Ю.В. Основи дискретної математики / Ю.В. Капітонова, С.Л. Кривий, О.А. Летичевський, Г.М. Луцький, М.К. Печурін. – К.: Наукова думка, 2002. – 580 с.

12. Кривий С.Л. Дискретна математика: Вибрані питання / С.Л. Кривий. – К.: Вид. дім «Києво-Могилянська академія», 2007. – 572 с.
13. Мальцев А.И. Алгоритмы и рекурсивные функции / А.И. Мальцев – М.: Наука, 1986. – 368 с.
14. Марков А.А. Теория алгоритмов / А.А. Марков, Н.М. Нагорный. – М.: Наука, 1984. – 432 с.
15. Мозговой М.В. Классика программирования: алгоритмы, языки, автоматы, компиляторы. Практический подход / М.В. Мозговой. – СПб.: Наука и Техника, 2006. – 320 с.
16. Нікольський Ю.В. Дискретна математика / Ю.В. Нікольський, В.В. Пасічник, Ю.М. Щербина. – К.: Видавнича група BHV, 2007. – 368 с.
17. Новиков Ф.А. Дискретная математика для програмистов / Ф.А. Новиков. – СПб.: Питер, 2001. – 304 с.
18. Оре. О. Приглашение в теорию чисел: Пер с англ. / О. Оре. – Изд. 2-е, стереотипное. – М.: Едиториал УРСС, 2003. – 128 с.
19. Пильщиков В.Н. Машина Тьюринга и алгоритмы Маркова. Решение задач / В.Н. Пильщиков, В.Г. Абрамов, А.А. Вылиток, И.В Горячая. – М.: МГУ, 2006. – 47 с.
20. Самохин А.В. Математическая логика и теория алгоритмов / А.В. Самохин. – Москва, 2003. – 237 с.
21. Тишин В.В. Дискретная математика в примерах и задачах / В.В. Тишин – СПб.: БХВ-Петербург, 2008. – 352 с.
22. Ding-Zhu Du. Problem Solving in Automata, Languages, and Complexity / Ding-Zhu Du, Ker-I Ko. – New York: WIP, 2001. – 388 p.
23. Salomaa A. Formal Languages / A. Salomaa. – New York: Academic Press, 1973. – 281 p.

Покажчик

Навчальне видання

Володимир Михайлович Гаврилків

ФОРМАЛЬНІ МОВИ ТА АЛГОРИТМІЧНІ МОДЕЛІ

Підписано до друку 12.09.2012. Формат 60×84/16.
Папір офсетний. Друк цифровий.
Гарнітура CM Roman. Умовн. друк. арк. 10,0.
Тираж 100. Зам. № 87 від 12.09.2012.

Віддруковано: Приватний підприємець О.М. Голіней
76000, м. Івано-Франківськ,
вул. Галицька,128,
тел.: (0342) 58-04-32.