МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет ім. М.Є. Жуковського
«Харківський авіаційний інститут»

І.В. Шостак, І.В. Груздо, М.О. Данова, Ю.І. Бутенко

ТЕОРІЯ АЛГОРИТМІВ І ОБЧИСЛЮВАЛЬНИХ ПРОЦЕСІВ

Навчальний посібник

Харків «ХАІ» 2013

УДК 510.51
Т33

Рецензенти: д-р техн. наук, проф. В.М. Левикін,
д-р техн. наук, проф. Є.І. Кучеренко

Теорія алгоритмів і обчислювальних процесів [Текст] : навч. посіб.
Т33 / І.В. Шостак, І.В. Груздо, М.О. Данова, Ю.І. Бутенко. - Х. : Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харк. авіац. ін-т», 2013. - 82 с.

Подано історичний екскурс для висвітлення етимології понять «алгоритм», «алфавітний оператор» і «алгоритмічні системи». Наведено основні засоби побудови алгоритмічних систем, що спрямовані на вирішення фундаментальних проблем теорії алгоритмів, а саме доказу обчислюваності та розв'язності. Особливу увагу приділено прикладним питанням математичної лінгвістики - організації ефективного машинного оброблення природно-мовних текстів з урахуванням семантики мікро- і макроконтексту.

Для студентів спеціальності 6.020303 «Прикладна лінгвістика».

Іл. 15. Табл. 16. Бібліогр. : 6 назв

УДК 510.51

ВСТУП

Теорія алгоритмів виникла у зв'язку із внутрішніми потребами теоретичної математики. Математична логіка, основи математики, алгебра, геометрія і аналіз залишаються дотепер однією з галузей застосування теорії алгоритмів.

Інша її галузь, що інтенсивно розвивається, виникла у зв'язку зі створенням цифрових обчислювальних і керуючих машин. Поява ЕОМ надала теорії алгоритмів яскраво вираженої прикладної спрямованості. Це, насамперед, алгоритмічні системи і алгоритмічні мови, що є основою сучасної теорії програмування для універсальних ЕОМ, і способи точного опису відображень, реалізованих цифровими автоматами.

Нарешті, прикладними аспектами сучасної теорії алгоритмів є ряд областей лінгвістики, економіки, фізіології мозку й психології, філософії, природознавства. Прикладом однієї із задач цієї області може бути точний опис алгоритмів, реалізованих людиною у процесі формування й прийняття рішень. У зазначеній області базис теорії алгоритмів використовується для створення засобами інженерії знань спеціальних методів підтримки прийняття рішень.

У посібнику також міститься матеріал, який охоплює весь комплекс питань створення, упровадження і експлуатації інтелектуальних засобів машинної обробки природно-мовних текстів з урахуванням їхньої семантики. Особливу увагу при цьому приділено технологіям пошуку інформаційних об’єктів у різного роду інформаційних середовищах, оскільки на цей час зазначені технології і програмні засоби, створені на їхній основі, є найбільш ефективними при реалізації завдань спрямованого пошуку інформації у репозиторіях значних обсягів.

1. АЛГОРИТМІЧНІ СИСТЕМИ

1.1. ІНТУЇТИВНЕ ПОНЯТТЯ АЛГОРИТМУ. ВЛАСТИВОСТІ АЛГОРИТМІВ

Під алгоритмом розуміють процес послідовного розв’язання задачі, що проходить у дискретному часі так, що в кожний наступний момент часу система об'єктів алгоритму формується за певним законом із системи об'єктів, що були в попередній момент часу. Строго кажучи, поняття алгоритму так само, як і поняття множини, математично точно визначити не можна.

Слово «алгоритм» походить, як припускають, від імені середньоазіатського (узбецького) математика IX ст. Аль Хорезми (Абу Абдалла Мухаммед ібн Муса аль Хорезми аль Меджусі) - «Algorithmi» у латинській транскрипції, який вперше сформулював правила (процедури) виконання чотирьох арифметичних дій у десятковій системі числення.

Поки обчислення були нескладними, особливої необхідності в алгоритмах не було. Коли ж з'явилася потреба в багаторазових покрокових процедурах, тоді й виникла теорія алгоритмів. Але при ще більшому ускладненні задач виявилося, що частину з них неможливо розв’язати алгоритмічним шляхом. Це - задачі, що розв'язуються людиною на основі міркувань. Розв’язання таких задач ґрунтується на використанні методів нейроматематики. У цьому випадку реалізуються процеси навчання, проб і помилок.

Якість алгоритму визначається його властивостями (характеристиками), основними з яких є такі.

1. Масовість. Передбачається, що алгоритм може бути придатний для розв’язання усіх задач даного типу. Наприклад, алгоритм для розв’язання системи лінійних алгебраїчних рівнянь має бути придатним для реалізації системи, що складається з довільної кількості рівнянь.

2. Результативність. Ця властивість означає, що алгоритм повинен приводити до одержання результату за скінченне число кроків.

3. Визначеність. Приписи, що входять у алгоритм, повинні бути точними й зрозумілими. Ця характеристика забезпечує однозначність результату обчислювального процесу при заданих вихідних даних.

4. Дискретність. Ця властивість означає, що описуваний алгоритмом процес і сам алгоритм можуть бути розбиті на окремі елементарні етапи, можливість виконання яких на ЕОМ у користувача не викликає сумнівів.

Може скластися враження, що за допомогою алгоритму можна розв’язати будь-які задачі. Виявляється, багато задач не можуть бути розв’язані алгоритмічно. Такі задачі називаються алгоритмічно нерозв'язними.

Для доказу алгоритмічної можливості розв’язності або нерозв'язності задач необхідні математично строгі й точні засоби. У середині 30-х років минулого століття були розпочаті спроби формалізації поняття алгоритму й запропоновані різні моделі алгоритмів: рекурсивні функції; «машини» Тьюринга, Поста; нормальні алгоритми Маркова.

Згодом було встановлено, що ці та інші моделі еквівалентні в тому розумінні, що класи розв'язуваних ними задач збігаються. Цей факт називається тезою Черча. Зараз це загальновизнано. Формальне визначення поняття алгоритму створило передумови для розроблення теорії алгоритму ще до розроблення перших ЕОМ. Прогрес обчислювальної техніки стимулював подальший розвиток теорії алгоритмів. Крім установлення алгоритмічної можливості розв'язання задач теорія алгоритмів займається ще й оцінюванням складності алгоритмів залежно від числа кроків (тимчасова складність) і необхідної пам'яті (просторова складність), а також займається розробленням ефективних у цьому сенсі алгоритмів.

Для реалізації деяких алгоритмів при будь-яких розумних з погляду фізики припущеннях про швидкість виконання елементарних кроків може знадобитися більше часу, ніж існує Всесвіт, на відміну від сучасних поглядів, або більше чарунок пам'яті, ніж атомів, що становлять планету Земля.

Існує ще одна задача теорії алгоритмів - вирішення питання виключення перебору варіантів у комбінаторних алгоритмах. Оцінювання складності алгоритмів й створення ефективних алгоритмів - одна з найважливіших задач сучасної теорії алгоритмів.

Ефективним прийнято вважати алгоритм, трудомісткість якого (число кроків) обмежена поліномом від характерного розміру задачі. Наведемо приклади ефективних і неефективних алгоритмів.

Жадібний алгоритм. Розглянемо скінченну множину X, що містить n елементів, деяку кількість його підмножин $X \subset 2^X$ і вагову функцію $w : X \to [0,+\infty)$.

Нехай

$$w(A) = \max_{B \in X} w(B),\ w(B) = \sum_{b \in B \subset X} w(b). \qquad (1.1)$$

Алгоритм, що у зазначеній кількості X вибирає підмножину A з максимальним значенням (вагою) $w(A)$, називається жадібним.

Жадібний алгоритм є ефективним; число його кроків становить $O(n)$.

Повний перебір. Дано скінченну множину $X = \{a_1,...,a_n\}$, що містить n елементів, і предикат $P(x_1,...,x_n)$. Цей предикат не є симетричним, тобто $P(...,x,...,y,...) \neq P(...,y,...,x,...)$. Потрібно знайти набір елементів $(a_{i_1},...,a_{i_n})$

такий, що $P(a_{i_1},...,a_{i_n}) = T$. Найпростіше розв'язання цієї задачі полягає у тому, що перебираються усі можливі перестановки $(a_{i_1},...,a_{i_n})$ з n елементів з перевіркою істинності предиката на них. Відомо, що число перестановок дорівнює $n!$. Отже, трудомісткість такого алгоритму повного перебору становить $O(n!)$. Оскільки $n!$ збільшується з ростом n швидше за будь-який поліном степеня n і швидше за 2^n, то даний алгоритм не є ефективним.

1.2. ФОРМАЛЬНІ КОНЦЕПЦІЇ СТРОГОГО ВИЗНАЧЕННЯ АЛГОРИТМІВ

При розроблені алгоритмів користуються одним із трьох основних методів.

Перший метод пов'язаний зі зведенням складної задачі до послідовності більш простих задач - така процедура називається методом приватних цілей. При цьому передбачається, що більш прості задачі легше піддаються обробленню, ніж первісні, а також, що розв'язання загальної задачі може бути отримане з розв'язання цих простих задач. Цей метод виглядає дуже розумно, але його не завжди легко перенести на конкретну задачу. Осмислений вибір більш простих задач - скоріше справа мистецтва або інтуїції, ніж науки. Більше того, не існує загального набору правил для визначення класу задач, які можна вирішити за допомогою такого підходу.

Другий метод розроблення алгоритмів відомий як метод підйому. Він починається із прийняття первісного припущення або обчислення початкового розв'язання задачі. Потім починається якомога швидкий рух "нагору" від початкового розв'язання в напрямку до кращих розв'язань. Коли алгоритм досягає такої точки, з якої неможливо рухатися нагору, він зупиняється. На жаль, ми не можемо ґарантувати, що остаточне розв'язання, отримане за допомогою алгоритму підйому, буде найкращим. Методи підйому запам'ятовують певну мету й намагаються зробити все, що можуть і де можуть, щоби якомога ближче наблизитись до мети. Це робить їх трохи недалекосяжними.

Третій метод відомий як відпрацювання назад, тобто починаємо з мети або розв'язання й рухаємося у зворотному напрямку до початкової постановки задачі. Потім, якщо ці дії обернені, рухаємося від постановки задачі до її розв'язання.

Евристичний алгоритм звичайно знаходить прийнятне, хоча не обов'язково оптимальне розв'язанням, його можна швидше й простіше реалізувати за будь-який відомий точний алгоритм. Багато евристичних алгоритмів ґрунтуються на методі приватних цілей або на методі підйому. Часто дуже якісні алгоритми мають розглядатися як евристичні, тобто

якщо ми побудували швидкий алгоритм, що працює на всіх відомих тестових задачах, але ми не можемо довести, що алгоритм правильний. Доки не надано такий доказ, алгоритм варто вважати евристичним.

Припис про послідовність дій алгоритму може бути поданий такими схемами: логічною, матричною, граф-схемою.

Логічну схему алгоритму (ЛСА) уперше було запропоновано радянським математиком О.М. Ляпуновим (1911 - 1973 рр.), професором кафедри математики військової артилерійської академії.

ЛСА - це вираз, що складається із символів операторів, логічних умов які йдуть у певному порядку, а також пронумерованих стрілок, розставлених певним чином.

Матрична схема алгоритму (МСА) - це квадратна матриця, елементи якої вказують умови передачі керування від і-го оператора рядка до j-го оператора стовпця. Рядки матриці нумеруються від першого оператора до передостаннього, стовпці - від другого до останнього.

Граф-схема алгоритму (ГСА) - це орієнтований граф особливого вигляду. Він містить вершини чотирьох типів: 1) операторні, позначувані прямокутниками; 2) умовні, позначувані ромбами; 3) початкову і 4) кінцеву вершини, позначувані овалами. Вершини з'єднуються дугами.

1.3. РЕКУРСИВНІ ФУНКЦІЇ

Рекурсія - один з основних прийомів програмування. Рекурсивні функції - функції, що залежать самі від себе. У теорії рекурсивних функцій, що вважається історично першою формалізацією поняття алгоритму, застосовується нумерація слів у будь-якому алфавіті натуральними числами (N), і будь-який алгоритм зводиться до обчислення деякої функції при цілочислових значеннях аргументів.

Функція є обчислюваною, якщо існує такий алгоритм, тобто покрокова процедура «від простого до складного», що за вхідним набором змінних обчислює значення функції, якщо цей вхідний набір належить до області визначення функції, або видає повідомлення про те, що вхідний набір не належить до області визначення функції.

Функція є напівобчислюваною, якщо при заданні вхідного набору, що не належить до області визначення функції, алгоритм не закінчує роботу (зациклюється). Теорію обчислюваності розробив А. Черч. Ідея була схожою на проблему функціональної повноти перемикальних функцій: виділити елементарні обчислювані функції (які «інтуїтивно обчислювані») - тобто базис - і запропонувати спосіб одержання із цих елементарних обчислюваностей функцій більше складних функцій за скінченне число кроків (як принцип суперпозиції в теорії перемикальних функцій). Отримані в такий спосіб функції теж будуть обчислені.

Такими елементарними обчислюваними функціями є:

$S(x) = x + 1$ - функція слідування (указує наступне натуральне число);

$O(x) = 0$ - нуль-функція;

$I_m^n(x_1, x_2, ..., x_n) = x_m$ – функції-проектори, результатами яких є вибірки m-го з n аргументів $1 \le m \le n$.

Для отримування з одних напівобчислюваних функцій інших функцій за скінченне число кроків використовують спеціальні оператори.

Першим з них є *оператор суперпозиції*, тобто підстановка у функцію функцій замість змінних. При цьому збільшується розмірність функції.

Визначення 1.1 (оператор суперпозиції). Будемо говорити, що n-місцева функція φ отримана з m-місцевої функції f і n-місцевих функцій $g_1, ..., g_m$ за допомогою оператора суперпозиції, якщо для усіх $x_1, ..., x_n$ справедлива рівність

$$\varphi(x_1, ..., x_n) = f(g_1(x_1, ..., x_n), ..., g_m(x_1, ..., x_n)). \qquad (1.2)$$

Другим оператором є *оператор примітивної рекурсії*.

Визначення 1.2. Говорять, що $(n+1)$-місцева функція φ отримана з n-місцевої функції f і $(n+2)$-місцевої функції g за допомогою оператора примітивної рекурсії, якщо для будь-яких $x_1, ..., x_n, y$ справедливі такі рівності:

$$\begin{aligned} &\varphi(x_1, ..., x_n, 0) = f(x_1, ..., x_n); \\ &\varphi(x_1, ..., x_n, y+1) = g(x_1, ..., x_n, y, \varphi(x_1, ..., x_n, y)). \end{aligned} \qquad (1.3)$$

Визначення 1.3 (оператор мінімізації). Будемо говорити, що n-місцева функція φ виходить із $(n+1)$-місцевих функцій f_1 і f_2 за допомогою оператора мінімізації або оператора найменшого числа, якщо для $x_1, ..., x_n, y$ рівність $\varphi(x_1, ..., x_n) = y$ справедлива тоді й тільки тоді, коли значення $f_i(x_1, ..., x_n, 0), ..., f_i(x_1, ..., x_n, y-1)$ $(i = 1, 2)$ знайдені і вони є нерівні попарно: $f_1(x_1, ..., x_n, 0) \ne f_2(x_1, ..., x_n, 0), ..., f_1(x_1, ..., x_n, y-1) \ne$ $\ne f_2(x_1, ..., x_n, y-1)$, а $f_1(x_1, ..., x_n, y) = f_2(x_1, ..., x_n, y)$.

Інакше, величина $\varphi(x_1, ..., x_n)$ дорівнює найменшому значенню аргумента y, при якому виконується остання рівність.

Розглянемо такий приклад задання числового ряду Фібоначчі 1,1, 2, 3, 5, 8, 13, 21, ... з використанням оператора примітивної рекурсії:

$$\begin{cases} f(0) = 1; \\ f(1) = 1; \\ f(n+2) = f(n) + f(n+1). \end{cases} \qquad (1.4)$$

Тут указуються два початкових значення функції f(0), f(1) і принцип формування наступного значення. Значення функції на деякому кроці на відміну від нульового й першого дорівнює сумі значень функції на двох

попередніх кроках.

Тоді $f(0)=1$, $f(1)=1$, $f(2)=f(0)+f(1)=1+1=2$, $f(3)=f(1)+f(2)=1+2=3$, $f(4)=f(2)+f(3)=2+3=5$, …

Чи всі функції є примітивно-рекурсивними? Можна показати, що множина всіх одномісних цілочислових функцій типу $N \mapsto N$, де N - множина натуральних чисел, незліченна. Тим більше це правильно для функції типу $N^n \mapsto N$. Кожна примітивно-рекурсивна функція має кінцевий опис, тобто задається кінцевим словом у деякому фіксованому для всіх функцій алфавіті. Множина всіх кінцевих слів є зліченною, тому примітивно-рекурсивні функції утворять не більш ніж зліченну підмножину незліченної множини функцій типу $N^n \mapsto N$. Однак виявляється, що не всі обчислювані функції можна описати як примітивно-рекурсивні.

Третім оператором є оператор мінімізації p, що дозволяє вводити в обчислення перебір для визначення потрібного значення.

Визначення 1.4. Функція називається примітивно-рекурсивною, якщо вона може бути отримана з найпростіших функцій O, S, I_m^n за допомогою скінченного числа застосувань операторів суперпозиції й примітивної рекурсії.

Приклад. Покажемо, що функція $s(x,y)=x+y$ може бути отримана з найпростіших функій за допомогою оператора примітивної рекурсії. Для цієї функції правильні тотожності

$$\begin{aligned} &x+0=x; \\ &x+(y+1)=(x+y)+1, \end{aligned} \tag{1.5}$$

які можна записати у вигляді

$$\begin{aligned} &s(x,0)=x; \\ &s(x,y+1)=s(x,y)+1, \end{aligned} \tag{1.6}$$

або

$$\begin{aligned} &s(x,0)=I_1^1(x); \\ &s(x,y+1)=S(s(x,y)). \end{aligned} \tag{1.7}$$

Це і є схема примітивної рекурсії, що ґрунтується на найпростіших функціях I_1^1 і S.

Визначення 1.5. Функція називається частково-рекурсивною, якщо вона може бути отримана з найпростіших функцій O, S, I_m^n за допомогою скінченного числа застосувань суперпозиції, примітивної рекурсії і μ-оператора. Якщо функція усюди визначена й частково-рекурсивна, то вона називається загально-рекурсивною.

Теорема 1.1 *(теорема Кліні)* [4]. Усяка частково-рекурсивна функція $f(x_1,...,x_n)$ може бути отримана за допомогою застосування μ-оператора і оператора суперпозиції з двох примітивно-рекурсивних функцій, з яких одна є раз і назавжди фіксованою, а інша залежить від функції $f(x_1,...,x_n)$.

Саме теорема Кліні свідчить про те, що існує така примітивно-рекурсивна функція U(x), що для усякої частково-рекурсивної функції $f(x_1,...,x_n)$ існує примітивно-рекурсивна функція $\theta(x_1,...,x_n,y)$ з такою властивістю:

$$f(x_1,...,x_n) = U(\mu y[\theta(x_1,...,x_n,y) = 0]). \qquad (1.8)$$

Зазначимо, що не завжди частково-рекурсивну функцію можна ефективно визначити як загально-рекурсивну. Ясно, що всяка примітивно-рекурсивна функція буде й частково-рекурсивною (і навіть загально-рекурсивною, тому що кожна примітивно-рекурсивна функція всюди визначена), оскільки для побудови частково-рекурсивних функцій з найпростіших використовується більше засобів, ніж для побудови примітивно-рекурсивних функцій. У той же час клас частково-рекурсивних функцій ширше за клас примітивно-рекурсивних функцій, оскільки всі примітивно-рекурсивні функції всюди визначені, а серед частково-рекурсивних функцій зустрічаються й функції не всюди визначені.

Поняття частково-рекурсивної функції виявилося цілком достатнім для формалізації поняття обчислюваної функції. При побудові аксіоматичної теорії висловлювань вихідні формули (аксіоми) і правила висновку вибиралися так, щоб отримані в теорії формули вичерпали б усі тавтології алгебри висловлень. До чого ж прагнемо ми в теорії рекурсивних функцій, чому саме так вибрали найпростіші функції й оператори для одержання нових функцій? Рекурсивними функціями ми прагнемо вичерпати всі мислимі функції, що піддаються обчисленню за допомогою будь-якої певної процедури механічного характеру. Подібно тезі Тьюринга у теорії рекурсивних функцій висувається відповідна природничо-наукова гіпотеза, що називаються *тезою Черча*.

Числова функція тоді й тільки тоді алгоритмічно (або машинно) обчислювана, коли вона є частково-рекурсивною.

Рекурсивні функції - основа функціонального програмування. Прикладом мови функціонального програмування є мова LISP, розроблена Д. Маккарті. Це одна з перших мов оброблення даних у символьній формі (LISP, від LISt Processing - оброблення списків). Однією з найбільш істотних властивостей мови LISP є те, що дані, програми й навіть сама мова є списками символів у дужках. Подібна структура дозволяє писати програми або підпрограми, які здатні звертатися самі до себе.

У LISP використовується префіксная форма запису:

$$(+xy) = x + y;$$
$$(*x(+xy)) = x * (x + y); \qquad (1.9)$$
$$(+(*xx)(*yy)) = x^2 + y^2.$$

Рекурсії присутні й у мовах структурного програмування.

1.4. АЛГОРИТМІЧНА КОНЦЕПЦІЯ ТЬЮРИНГА. ОБЧИСЛЮВАНІСТЬ ЗА ТЬЮРИНГОМ

Теза Тьюринга (основна гіпотеза теорії алгоритмів) [3].

Повернемося до інтуїтивного подання поняття алгоритму. Нагадаємо, що одна із властивостей алгоритму полягає у тому, що він являє собою єдиний спосіб, який дозволяє для кожної задачі з якоїсь нескінченної множини задач за скінченне число кроків знайти її розв'язання. На поняття алгоритму можна глянути також з іншого боку. Кожну задачу з нескінченної множини задач можна виразити (закодувати) деяким словом деякого алфавіту, а розв'язання задачі - якимось іншим словом того ж алфавіту. У результаті цього одержимо функцію, задану на деякій підмножині множини всіх слів обраного алфавіту й такого, що набуває значення в множині всіх слів того ж алфавіту. Розв'язати будь-яку задачу - значить, знайти значення цієї функції на слові, що кодує дану задачу. А мати алгоритм для розв'язання усіх задач даного класу - значить, мати єдиний спосіб, що дозволяє в скінченне число кроків «обчислювати» значення побудованої функції для будь-яких значень аргументу з її області визначення. Таким чином, алгоритмічна проблема - це, по суті, проблема про обчислення значень функції, заданої у деякому алфавіті.

Залишається уточнити, що значить уміти обчислювати значення функції. Це означає, що значення функції обчислюють за допомогою відповідної машини Тьюринга. Для яких же функцій можливо їх тьюринговое обчислення? Численні дослідження вченими даної проблеми показали, що такий клас функцій надзвичайно широкий. Кожна функція, для обчислення значень якої існує будь-який алгоритм, виявлялася обчислюваною за допомогою деякої машини Тьюринга. Це дало привід Тьюрингу висловити таку гіпотезу, що називається основною гіпотезою теорії алгоритмів, або тезою Тьюринга.

Для знаходження значень функції, заданої у деякому алфавіті, тоді й тільки тоді існує будь-який алгоритм, коли функція є обчислюваною за Тьюрингом, тобто коли вона може обчислюватися на відповідній машині Тьюринга.

Це означає, що строго математичне поняття обчислюваності (за Тьюрингом) функції є ідеальною моделлю взятого з досвіду поняття алгоритму. Дана теза є не що інше, як аксіома, постулат, висунутий нами, про взаємозв'язки нашого досвіду з тією математичною теорією, яку ми під цей досвід хочемо підвести. Звичайно ж дана теза не може бути доведена

методами математики, оскільки вона не має внутрішньоматематичного характеру (одна сторона в тезі - поняття алгоритму - не є точним математичним поняттям). Вона висунута виходячи з досвіду, і саме досвід підтверджує її обгрунтованість. Точно так само, наприклад, не можуть бути доведені й математичні закони механіки; вони відкриті Ньютоном і багаторазово підтверджені досвідом.

Утім, не виключається принципова можливість того, що теза Тьюринга буде спростована. Для цього повинна бути зазначена функція, яка є обчислюваною за допомогою будь-якого алгоритму, але не є обчислюваною на будь-якій машині Тьюринга. Така можливість є малоймовірною (у цьому - одне зі значень гіпотези): усякий алгоритм, що буде відкритий, може бути реалізований на машині Тьюринга.

Додаткові непрямі доводи на підтвердження цієї гіпотези будуть наведені далі, де розглядаються інші формалізації інтуїтивного поняття алгоритму й доводиться їхня рівносильність із поняттям машини Тьюринга.

Машина Тьюринга

Машина Тьюринга [1] - одна з абстрактних моделей алгоритмів, названа на честь англійського математика Алана Тьюринга (1912 - 1954 рр.). Машина Тьюринга містить:

1) керуючий пристрій, що може перебувати в одному зі станів, що утворюють скінченну множину $Y = \{y_1, y_2, ..., y_k\}$;

2) стрічку, розбиту на чарунки, у кожній з яких може бути записаний один із символів скінченного алфавіту $X = \{x_1, x_2, ..., x_n\}$;

3) пристрій звертання до стрічки, що зчитує й записує інформацію, яка у кожний момент часу «обдивляється» чарунку і залежно від символу в чарунці й стану керуючого пристрою записує у цю чарунку новий символ (він може збігатися з колишнім, зчитувальним) або порожній символ (колишній символ стирається і на його місце записується символ пробілу). Далі пристрій обігу зсувається на чарунку уліво чи вправо або залишається на місці. При цьому керуючий пристрій переходить у новий внутрішній стан або залишається в старому (поточному) стані. Серед станів пристрою керування виділяються початковий y_1 і заключний y_k стани (k - мнемонічний знак закінчення роботи). У початковому стані машина Тьюринга перебуває перед початком роботи, у скінченному - після закінчення роботи.

Пам'ять машини Тьюринга - скінченна множина станів (внутрішня пам'ять) і стрічка (зовнішня пам'ять). Стрічка нескінченна в обидва боки, однак у початковий момент часу тільки скінченне число чарунок стрічки заповнено символами, інші порожні, тобто містять порожній символ - символ пробілу?

Дані машини Тьюринга - слова в алфавіті стрічки, на стрічці записуються вихідні дані, а потім - результат.

Елементарні кроки - це зчитування й запис символів, зрушення головки, перехід пристрою керування зі стану в стан.

Під впливом вхідного символу X, наприклад 1, що зчитується зазначеним вище пристроєм, система керування формує вихідний символ Z, управляє рухом головки вліво (L), вправо (R), на місці (E).

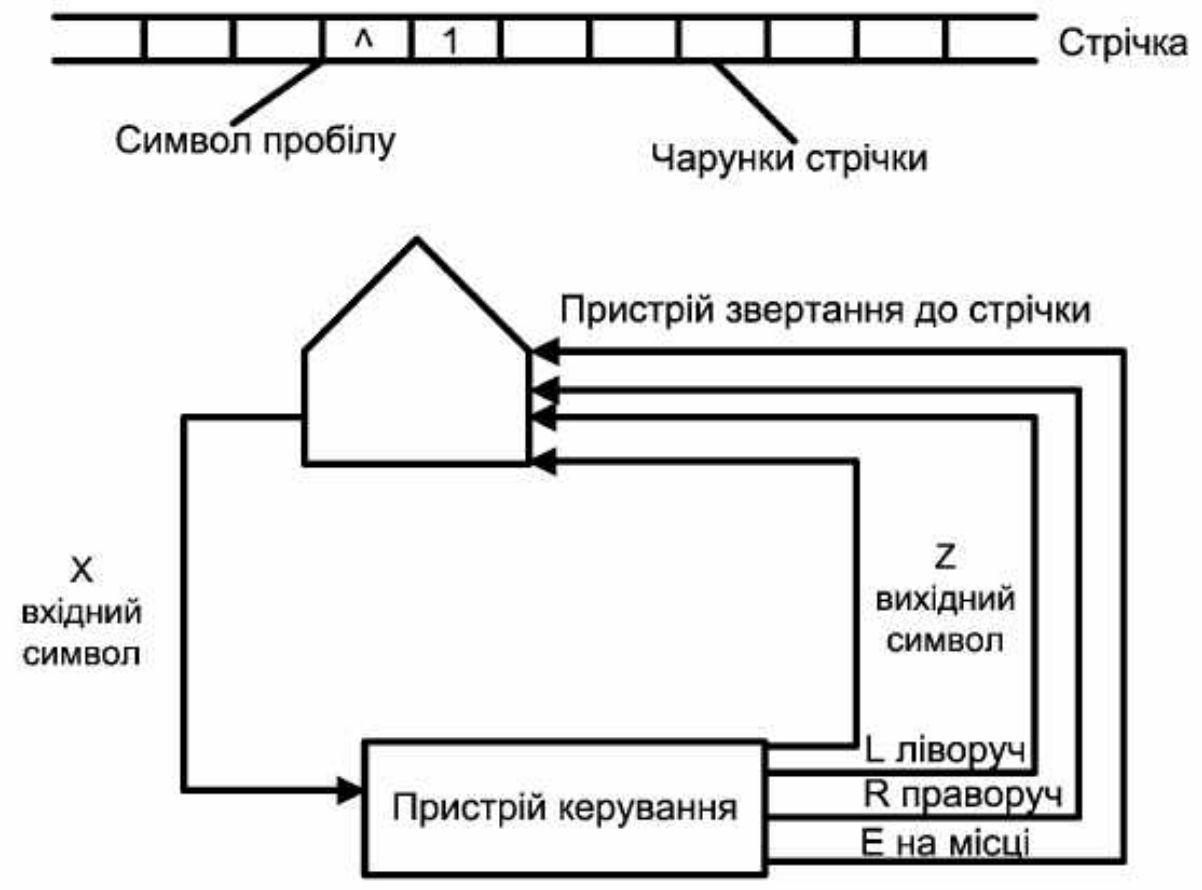


Рис. 1.1. Структурна схема машини Тьюринга

Повний стан машини, або конфігурація (або машинне слово), за яким однозначно визначається її подальше поводження, описується внутрішнім станом y_i, символами ліворуч і праворуч від пристрою, наприклад: $...x_1 y_i x_2...$ Система команд машини містить записи вигляду $y_1 x_1 \to \frac{y_2}{zR}$, де x_1 - символ у стані y_1, що читається; y_2 - новий внутрішній стан; z - символ, що записується; R - ознака просування пристрою; $\to$ - символ переходу в новий стан.

Сукупність усіх команд називається програмою. Кожна машина Тьюринга визначається своїм алфавітом, станами внутрішньої пам'яті й програмою. Щоб повністю визначити роботу машини, треба вказати її конфігурацію для початкового моменту часу. Будемо вважати, що в початковій конфігурації головка сприймає саму ліву непорожню чарунку.

Машина Тьюринга задається трійкою даних - $M =< X, Y, П >$, де X - алфавіт символів стрічки з виділеним порожнім символом λ (пробілом);

Y - алфавіт внутрішніх станів з виділеними символами початкового y_1 і заключного y_k станів; *П* - програма, тобто кінцева послідовність упорядкованих п'ятірок символів - команд.

Якщо машина, почавши роботу з деяким словом, записаним на стрічці, приходить у заключний стан, то вона називається застосовною до цього слова. Результатом її роботи вважається слово, записане на стрічці в заключному стані. Якщо ж машина у кожен момент часу не приходить у заключний стан, то вона називається непридатною до даного слова й результат її роботи є невизначеним.

Обчислювані за Тьюрингом функції

Функція називається обчислюваною за Тьюрингом, якщо існує машина Тьюринга, що обчислює її, тобто така машина Тьюринга, що обчислює її значення для тих наборів значень аргументів, для яких функція визначена, і така, яка працює вічно, якщо функція для даного набору значень аргументів не є визначена [1, 3].

Часткова числова n-місцева функція $f(x_1,...,x_n)$ називається обчислюваною за Тьюрингом, якщо існує машина, що обчислює її за таких умов.

1. Якщо набір значень аргументів належить до області визначення функції f, то машина, почавши роботу в деякій конфігурації, що задає значення аргументів, зупиняється, закінчуючи роботу у конфігурації, що відповідає значенню функції.

2. Якщо набір значень аргументів не належить до області визначення функцій f, то машина, почавши роботу в деякій конфігурації, працює нескінченно, не приходячи у заключний стан.

1.5. НОРМАЛЬНИЙ АЛГОРИТМ МАРКОВА. ОБЧИСЛЮВАНІСТЬ ЗА МАРКОВИМ

Теорія нормальних алгоритмів (або алгорифмів, як називав їх творець теорії) була розроблена радянським математиком А. А. Марковим (1903 - 1979) наприкінці 1940-х - на початку 1950-х рр. XX ст. Ці алгоритми являють собою деякі правила з переробки слів у будь-якому алфавіті так, що вихідні дані й шукані результати для алгоритмів є словами в деякому алфавіті.

Марковські підстановки. Алфавітом (як і колись) називається будь-яка непуста множина. Його елементи називаються буквами, а будь-які послідовності букв - словами в даному алфавіті. Для зручності міркувань допускаються порожні слова (вони не мають у своєму складі жодної букви). Порожнє слово будемо позначати $\wedge$. Якщо A і B - два алфавіти, причому $A \subseteq B$, то алфавіт B називається розширенням алфавіту A.

Слова будемо позначати латинськими буквами P, Q, R (або цими ж буквами з індексами). Одне слово може бути складовою частиною іншого

слова. Тоді перше називається підсловом другого, або входженням у друге. Наприклад, якщо А - алфавіт російських букв, то можемо розглянути такі слова: *Р1* = параграф, *Р2* = граф, Р3 = ра. Слово *Р2* є підсловом слова *Р1*, а *Р3* - підсловом слів *Р1* і *Р2*, причому в *Р1* воно входить двічі. Особливий інтерес становить перше входження.

Визначення 1.6. Марковською підстановкою [3] називається операція над словами, що задається за допомогою упорядкованої пари слів *(P, Q)*, що полягає у такому. У заданому слові R знаходять перше входження слова *P* (якщо таке є) і, не змінюючи інших частин слова R, замінюють у ньому це входження словом Q. Отримане слово називається результатом застосування марковської підстановки *(P, Q)* до слова *R*. Якщо ж першого входження P у слово R немає (і, отже, взагалі немає жодного входження *P* у *R*), то вважається, що марковська підстановка *(P, Q)* не може бути застосовна до слова R.

Для позначення марковської підстановки (P, Q) використовується запис $P \to Q$, який називається формулою підстановки (P, Q). Деякі підстановки *(P,Q)* будемо називати заключними (зміст назви стане зрозумілим трохи пізніше). Для позначення таких підстановок будемо використовувати запис $P \to Q$, називаючи його формулою заключної підстановки. Слово *P* називається лівою частиною, а *Q* - правою частиною у формулі підстановки.

Упорядкований кінцевий список формул підстановок в алфавіті *A* називається схемою (або записом) нормального алгоритму в *A*. (Запис точки в дужках означає, що вона може стояти у цьому місці, а може бути відсутня). Ця схема визначає (детермінує) алгоритм перетворення слів, який називається нормальним алгоритмом Маркова. Дамо його точне визначення.

Визначення 1.7. Нормальним алгоритмом (Маркова) в алфавіті *A* називається таке правило побудови послідовності V_i слів в алфавіті *A*, яка виходить з даного слова *V* у цьому алфавіті. Як початкове слово V_0 послідовності береться слово *V*. Нехай для деякого $i > 0$ слово V_i побудоване й процес побудови розглянутої послідовності ще не завершився. Якщо при цьому в схемі нормального алгоритму немає формул, ліві частини яких входили б у V_i, то V_{i+1} будуть такими, що дорівнюють V_i, і процес побудови послідовності вважається таким, що завершився. Якщо ж у схемі є формули з лівими частинами, що входять у V_i, то як V_{i+1} береться результат марковської підстановки правої частини першої з таких формул замість першого входження її лівої частини в слово V_i. Процес побудови послідовності вважається таким, що завершився, якщо на даному кроці була застосована формула заключної підстановки, і триваючої - у противному разі. Якщо процес побудови згаданої

послідовності обривається, то говорять, що розглянутий нормальний алгоритм застосовний до слова V. Останній член непослідовності називається результатом застосування нормального алгоритму до слова V. Говорять, що нормальний алгоритм переробляє V у W.

Послідовність V_i будемо записувати так:

$$V_0 \Rightarrow V_1 \Rightarrow V_2 \Rightarrow ... \Rightarrow V_{m-1} \Rightarrow V_m, \qquad (1.10)$$

де $V_0 = V$ і $V_m = W$.

Ми визначили поняття нормального алгоритму в алфавіті A. Якщо ж алгоритм задано у деякому розширенні алфавіту A, то говорять, що він є нормальним алгоритмом над A.

Нормально обчислювані функції і принцип нормалізації Маркова

Як і машини Тьюринга, нормальні алгоритми не виконують самих обчислень: вони лише перетворюють слова, замінюючи в них одні букви іншими за запропонованими їм правилами. У свою чергу, ми пропонуємо їм такі правила, результати застосування яких ми можемо інтерпретувати як обчислення.

Визначення 1.8. Функція f, задана на деякій множині слів алфавіту A, називається нормально обчислюваною, якщо знайдуться такі розширення даного алфавіту ($A \subseteq B$) і такий нормальний алгоритм у B, що кожне слово V (в алфавіті A) з області визначення функції f перетворює цей алгоритм у слово $f(V)$ [3].

Творець теорії нормальних алгоритмів А. А. Марков висунув гіпотезу, що одержала назву *принцип нормалізації Маркова*. Відповідно до цього принципу для знаходження значень функції, заданої у деякому алфавіті, тоді й тільки тоді існує будь-який алгоритм, коли функція є нормально обчислюваною.

1.6. МЕТОДИ ОЦІНЮВАННЯ АЛГОРИТМІВ

У теорії алгоритмів поняття «алгоритм» зазвичай уточнюється за допомогою опису «математичної моделі» обчислювальної машини. Тут можливі два підходи залежно від того, чи оцінюється складність алгоритму (машини, програми) або здатність обчислювального процесу, що протікає відповідно до алгоритму.

Для обґрунтованого використання часу й пам'яті обчислювальної машини необхідний теоретичний і емпіричний аналіз ефективності.

На практиці доводиться задовольнятися наближеним розв’язанням задачі й миритися з елементами невизначеності у розв’язанні. Як правило, точно розв’язуються тільки задачі зі скінченним числом вхідних і вихідних даних, що допускають скінченне подання. Можна виділити дві причини, за якими обмежуються наближеним розв’язанням: або задачу неможливо

розв’язати точно, або точне розв’язання не потрібно. Неможливість одержати точне розв’язання може пояснюватися тим, що:

- інформація неповна, тобто деякі незбіжні між собою елементи задачі не можна розрізнити, маючи у своєму розпорядженні тільки цю інформацію;
- інформація наближена, така ситуація може з'явитися як наслідок багатьох причин, у тому числі помилок ЕОМ, помилок при передачі даних, обмеженої точності подання й оброблення чисел, обмежень на точність вимірів;
- обмежено клас припустимих алгоритмів.

Більшість реальних задач доводиться розв’язувати, маючи у своєму розпорядженні лише неповну або наближену інформацію. Чисельні процедури використовують арифметику кінцевої точності й ґрунтуються на теорії апроксимації. У принципі бажано описувати числові алгоритми з тією же строгістю, що й алгебричні. Поняття обчислення може відноситися не тільки до чисел. Перші символьні маніпуляції були пов'язані з використанням шифрів і тайнопису. Протягом півстоліття поширюються програмні системи для формульних перетворень. Іншим прикладом символьних обчислень є робота з текстами, ігри в шашки, шахи, го. До цієї ж групи програм належать програми для одержання математичних доказів.

Алгебричні алгоритми реалізуються у програмних системах, що припускають уведення й висновок інформації в символьних позначеннях. Вона відрізняються простими формальними описами, існуванням доказів правильності й асимптотичних меж часу виконання. Крім того алгебричні об'єкти можна точно подати в пам'яті ЕОМ, тому алгебричні перетворення можуть бути виконані без втрати точності й значущості. Точність при використанні алгебричного алгоритму часто потребує більшого часу виконання й необхідного об'єму пам'яті, ніж для його числового аналога.

Існує ряд важливих причин для аналізу алгоритмів. Однією з них є необхідність отримання оцінок (або границь) для об'єму пам'яті або часу роботи, що буде потрібно алгоритму для успішного оброблення конкретних даних. Машинний час і пам'ять - це ресурси відносно дефіцитні й дорогі. Інша причина - бажання мати якийсь кількісний критерій для порівнянь двох алгоритмів, що претендують на розв’язання однієї й тієї ж задачі. Ще одна причина - бажання мати механізм для виявлення найбільш ефективних алгоритмів. Іноді неможливо скласти чітку думку про відносну ефективність двох алгоритмів. Один з них може в середньому краще працювати, наприклад, на випадкових вхідних даних, інший краще працює на якихось спеціальних даних. Важливо також установити абсолютний критерій. Необхідно вважати розв’язання задачі оптимальним тоді, коли наш алгоритм є ефектним і його неможливо (незалежно від наших розумових здатностей) більш поліпшити.

Зазвичай від алгоритму потрібна ефективність. Це означає, що всі

операції, які необхідно виконати в алгоритмі, мають бути досить простими, щоб їх у принципі можна було виконати точно й за короткий проміжок часу за допомогою олівця й паперу.

Обмеження фінітності для практичних цілей є недосить твердим: використовуваний алгоритм повинен мати не просто скінченне, а гранично скінченне, розумне число кроків. У реальних обчисленнях головне питання щодо деякої функції полягає не в тому, чи "обчислювана дана функція", а скоріше за все в тому, чи " обчислювана вона практично". Тобто чи існує програма, що обчислює функцію за час, на який ми розраховуємо. Можна вимірювати час, необхідний для обчислення кожного значення функції за конкретною програмою, допускаючи, що кожний крок відбувається за одиницю часу, а мірою обчислювальної складності брати час обчислення.

Недостатньо довести правильність алгоритму. Усі ми можемо зробити помилки при доведенні й при перекладі правильного алгоритму в програму. Кожний може забути деякий окремий випадок задачі. Деякі особливості операційної системи можуть спричинити для деяких вхідних даних такі дії частини Вашого алгоритму, про які Ви й не підозрювали. Програма повинна бути перевірена для широкого спектра припустимих вхідних даних. Цей процес може бути стомлюваним і складним. Аналітичний і експериментальний аналіз доповнюють один одного. Аналітичний аналіз може бути неточним, якщо зроблені занадто сильні спрощувальні припущення. У цьому випадку можуть бути отримані тільки грубі оцінки. Експериментальні результати, особливо коли використовуються випадково згенеровані дані, можуть виявитися занадто однобічними. Щоб одержати достовірні результати, варто проводити як аналітичне, так і експериментальне дослідження там, де це можливо.

Як міра складності алгоритмів розглядається функціонал, що співвідносить кожному алгоритму A деяке число $\mu(A)$, що характеризує (у підходящому змісті) його громіздкість, наприклад: число команд в A, довжина запису A або будь-який інший числовий параметр, що характеризує обсяг інформації, що втримуються в A. Подібні функціонали вже давно застосовуються в теоретико-кібернетичних дослідженнях схем, що реалізують функції алгебри логіки (Шеннон, Яблонський, Ляпунов та ін.), а також в обчислювальній математиці, де потужність схеми, за якою обчислюється багаточлен, вимірюється числом арифметичних операцій, що фігурують у схемі. Розходження полягає лише в тому, що в зазначених роботах розглядаються спеціальні вузькі класи функцій і спеціальні способи опису алгоритмів. Ми ж зацікавлені в розробленні й застосуванні аналогічних понять у більш загальній ситуації (будь-які обчислювані функції, загальні концепції алгоритму). Перші публікації, у яких такі міри складності знайшли застосування, належать А. А. Маркову й А. Н. Колмогорову.

Як міра складності обчислень розглядається функціонал, що

співвідносить кожній парі *(A, a)*, де *A* — алгоритм; *a* — індивідуальна задача з того класу задач, які алгоритм А вирішує, $\delta(A,a)$ - деяке число**.** Це число характеризує складність роботи алгоритму А відповідно до вихідних даних *a* до видачі відповідного результату. Наприклад, як $\delta(A,a)$ можна брати число елементарних кроків, з яких складається ця робота (інакше кажучи, тривалість процесу обчислення) або об'єм пам'яті, що може знадобитися для реалізації усіх викладок по ходу даного процесу, і т.д. Оскільки для кожного алгоритму *A* однозначно визначений той клас задач Ω, що він вирішує (наприклад, та функція *f***,** значення якої він обчислює), то можна вважати, що в розглянутій ситуації кожний алгоритм *A* характеризується функцією змінного $\alpha\delta_A(a) \underset{df}{=} \delta(A,a)$. Іншими словами, мірою складності роботи алгоритму (обчислення) є оператор, що зіставляє з кожним алгоритмом *A* відповідну функцію $\delta_A(a)$.

Такий підхід до оцінювання складності обчислень належить радянському математику Г. С. Цейтині. За Г. С. Цейтиною, складність роботи нормального алгоритму вимірюється функцією, що вказує залежність числа кроків алгоритму від слова, до якого він застосовується. Приблизно в той же час і незалежно від Г. С. Цейтини Б. А. Трахтенброт увів аналогічні функції, що вимірюють об'єм пам'яті, необхідної для рекурсивних обчислень, і назвав їх сигналізувальними функціями.

Визначаючи певну міру складності для алгоритмів (або обчислень), ми тим самим сподіваємося одержати зручний інструмент для порівняння алгоритмів, а також для оцінювання об'єктивних труднощів, властивого різним обчислюваним функціям. У зв'язку з цим можна зазначити розходження у здійсненні такого задуму залежно від того, чи розглядається міра складності алгоритму або міра складності його роботи. Оскільки складність алгоритму вимірюється дійсним числом, то будь-які два алгоритми порівнянні за складністю і дорівнюють один одному.

Зазвичай вважають, що міра складності алгоритму набуває лише натурального значення, тому для кожної обчислюваної функції існує її обчислювальний алгоритм, з мінімальною складністю. Цю мінімальну складність природно розглядати як складність самої функції, тим самим і множина всіх залежних функцій упорядкована за ступенем складності цих функцій.

Якщо ж вихідною є міра складності обчислень, то виходить інша картина. Дві сигналізувальні функції можуть виявитися непорівнянними, навіть якщо вважати, як зазвичай це прийнято, що сигналізувальна δ_A менше ***сигналізувальної*** δ_B, якщо для усіх α може бути кінцеве число $\delta_A(\alpha) < \delta_B(\alpha)$. Тому як тільки зафіксована деяка функція *f***,** то апріорі не ясно, чи існує для неї найкраще обчислення. Тут доводиться обмежуватися більш слабкою характеристикою складності самої функції *f***,**

а саме відшукуються дві функції φ_1 (нижня оцінка) і φ_2 (верхня оцінка), такі, що:

1) існує алгоритм A_1, який обчислює f з сигналізувальною, яка не перебільшує φ_2;

2) яким би не був алгоритм A, *що* обчислює f, його сигналізувальна не менше φ_1.

Зрозуміло, чим ближче одна до одної верхня й нижня оцінки φ_1 і φ_2, тим точніше охарактеризована складність самої функції f.

Отже, на відміну від ієрархій, основаних на складності алгоритмів, ієрархії, основані на мірі складності обчислень, є частково впорядкованими. Це ускладнює їх вивчення, але разом з тим дозволяє, очевидно, більш тонко вловлювати сутність того, що ми інтуїтивно розуміємо під складністю обчислень функції.

Теорія складності залежить від покладених у її основу концепцій алгоритму (рекурсивні функції, машини Тьюринга тощо), а також від обраної міри складності.

Тому апріорі можна припустити, що при переході від одних концепцій алгоритму до інших необхідно заново будувати теорію складності. Однак ідея моделювання одних алгоритмів іншими рятує нас від цього.

Розглянемо оцінювання складності алгоритму стосовно машин Тьюринга.

З кожною конфігурацією k, до якої може бути застосована машина Тьюринга, можна асоціювати число, що характеризує складність процесу $M(k)$ у тому або іншому сенсі.

Варіюючи k, одержимо функцію від неї, визначену на множині всіх конфігурацій, до яких дана машина може бути застосовна. Функції такого роду ми називаємо ***сигналізувальними функціями.***

Роботу машини M будемо характеризувати такими сигналізувальними функціями:

Сигналізувальна часу $S_M(k)$ дорівнює тривалості процесу $M(k)$ (тобто числу його конфігурацій), якщо M може бути застосовна до k, і $t_M(k)$ не визначена, якщо M може бути не застосовна до k.

Активною зоною процесу називається мінімальна частина стрічки, що обхоплює активні зони всіх його конфігурацій. Робочою (активною) зоною даної конфігурації називається мінімальна зв'язана частина стрічки, що містить оглядову чарунку, а також усі чарунки, у яких записані значущі букви.

Сигналізувальна ємність $S_M(k)$ дорівнює довжині активної зони процесу $M(k)$, якщо M застосовна до k, і не визначена - у противному разі.

Сигналізувальна коливань $\omega_M(k)$ дорівнює числу коливань (зміни напрямків) зчитуючого пристрою за час $t_M(k)$, якщо M застосовна до k, і не визначена - у протилежному випадку.

Сигналізувальна режиму $r_M(k)$ дорівнює максимальному числу проходжень головки через край чарунки за час $t_M(k)$, якщо M застосовна до k і не визначена - у противному випадку.

Зокрема, коли як конфігурації k взяті початкові конфігурації для слова p, то процес буде характеризуватися ***сигналізувальними*** функціями $t_M(k), S_M(k), \omega_M(k), r_M(k)$. Поряд з ними розглядаються і функції типу

$$t_M^{(n)} = \max_{|p|=n} t_M(p), \quad \omega_M^{(n)} = \max_{|p|=n} \omega_M(p);$$

$$S_M^{(n)} = \max_{|p|=n} S_M(p), \quad r_M^{(n)} = \max_{|p|=n} r_M(p),$$

де $|p|$ — довжина слова p, а також функції типу

$$t_M^*(n) = \max_{\nu \le n} t_M(\nu), \quad \omega_M^*(n) = \max_{\nu \le n} \omega_M(\nu);$$

$$S_M^*(n) = \max_{\nu \le n} S_M(\nu), \quad r_M^*(n) = \max_{\nu \le n} r_M(\nu),$$

де ν — найбільше з довжин слів p_i.

Побудова машини дає лише верхні сигналізувальні оцінки. Знаходження ж нижніх оцінок є більш складною справою й потербує спеціальної теорії.

Має місце теорема, яка показує, що за наявності верхньої оцінки для будь-якої однієї із сигналізувальних t, s, ω, r і заданих параметрів m і n (m — число символів зовнішнього алфавіту, n — число символів внутрішнього алфавіту) можна одержати верхні оцінки для інших.

Теорема 1.2. Для будь-якої машини M і для будь-якого слова p, до якого вона застосовна, справедливі такі нерівності:

$$\omega(p) \le t(p);$$

$$r(p) \le \omega(p) + 1;$$

$$s(p) \le |p| + 2n^{r(p)+1};$$

$$t(p) \le m^{s(p)} \cdot s(p) \cdot n.$$

Розглянемо більш докладно оцінювання **складності алгоритму P і Np-класи складності**.

Складність - це спосіб порівняння алгоритмів [1]. Їх порівнюють за кількістю необхідних для виконання алгоритму кроків (тимчасова складність) і за обсягом пам'яті, необхідним для роботи

алгоритму (ємнісна складність).

Складність алгоритму відображує витрати, необхідні для його роботи. Будемо розглядати тимчасову складність. Це функція, яка кожній вхідній довжині n ставить у відповідність мінімальний час, що затрачується алгоритмом на розв'язання усіх однотипних індивідуальних задач цієї довжини.

З курсу математичного аналізу відомо, що функція $f(n)$ є $O(g(n))$, якщо існує константа c така, що $|f(n)| \le c(g(n))$ для всіх $n \ge 0$, де $O(n)$ – складність порядку n; n – параметр вихідних даних алгоритму; $O(n^2)$ – складність порядку n^2.

Складність буває мінімальною, середньою й максимальною.

Класи задач P і NP

Складність задачі оцінюють, як правило, з точки зору витрати часу, необхідного машині Тьюринга - Поста для того, щоб обчислити функцію, за допомогою якої знаходяться розв'язання розглянутої задачі [1, 2].

Клас задач P

Розглянемо масову задачу, під якою в дійсності мають на увазі клас Π однотипних задач, що складається з нескінченного числа індивідуальних конкретних задач і описується сукупністю параметрів. Кожній конкретній задачі $Z \in \Pi$ відповідає свій набір параметрів. До кожної задачі ставиться запитання, відповідь на яке має вигляд «ТАК» чи «НІ». Наприклад, нас цікавить, чи мають частково-рекурсивні функції властивість Х.

Будемо вважати, що розв'язання задачі Π зводиться до обчислення на машині Тьюринга - Поста деякої функції.

Припустимо, що з класом П позв'язана система кодування α, яка кожній задачі Z ставить у відповідність слово $\alpha(Z)$ у деякому алфавіті. Розмір задачі Z – це довжина слова $|\alpha(Z)|$.

Нехай машина Тьюринга T розв'язує задачі класу Π і класу

$$t_T(n) = \max_{Z,|\alpha(Z)|=n} t_T(\alpha(Z)), \qquad (1.11)$$

що є відповідно тимчасовою складністю (у гіршому випадку).

Говорять, що машина Тьюринга T розв'язує задачу класу Π за поліноміальний час, якщо

$$t_T(n) = O(p(n)) \qquad (1.12)$$

для деякого полінома $p(n)$. У противному разі говорять, що задача розв'язується за експонентний час.

Клас задач Π називається полиномільно розв'язним, якщо існує машина Тьюринга T, що розв'язує задачі $z \in \Pi$ за поліноміальний час.

Сукупність класів задач, що розв'язуються за поліноміальний час, називається класом задач P.

Клас задач NP

Масова задача Π належить до класу NP, якщо у випадку відповіді «ТАК» для задачі $z \in \Pi$ існує слово $c(Z)$ довжиною $O(p(|\alpha(Z)|))$ таке, що задача $(Z, c(Z))$ належить до класу P. Слово $c(Z)$ називається посвідченням (або здогадкою) задачі Z. Його наявність дозволяє перевірити приналежність задачі Z до класу P.

Наприклад, можливість виконання формули $A(X_1,...,X_n)$ перевіряється, якщо крім самої формули даний конкретний набір змінних $x_1^0,...x_1^0$ – припущення, що сприяє установленню здійсненності формули A.

1.7. АЛГОРИТМІЧНО РОЗВ'ЯЗНІ Й НЕРОЗВ'ЯЗНІ ПРОБЛЕМИ

Об'єктом вивчення у теорії алгоритмів є, перш за все, алгоритмічна розв'язність деякої масової задачі. Розв’язувана задача та, для якої є абстрактна модель, що за скінченне число кроків перевіряє для будь-яких вхідних даних, чи є розв’язання даної задачі.

Прикладами алгоритмічно розв'язних проблем є [1]:

- знаходження суми двох чисел;
- можливість розв'язання пропозиційних формул, тобто знаходження алгоритму, що дозволяє для будь-якої ПФ за скінченне число дій вирішити, чи є вона тавтологією, чи ні;
- рівносильність двох пропозиційних формул, тобто проблема знаходження алгоритму, що з'ясовує рівносильність або нерівносильність двох будь-яких заданих ПФ, та ін.

Проблеми називають алгоритмічно нерозв'язними, якщо не існує єдиного алгоритму, що вирішує усі індивідуальні задачі цих проблем. Це не значить, що такого алгоритму не існує для деякого підкласу всього класу задач алгоритмічно нерозв'язної проблеми. Наприклад, проблема розв’язання предикатних формул, проблема рівносильності двох предикатних формул, проблеми застосовності для машин Тьюринга й нормальних алгоритмів Маркова, проблема еквівалентності слів в асоціативних численнях та інші алгоритмічно нерозв'язними.

Нагадаємо, що машина називається застосовною до початкового слова, якщо вона, почавши працювати з цим словом, прийде в заключний стан.

Приклад непридатної машини - машина Тьюринга, у якої у першій частині команд не зустрічається заключний стан y_k. Машина M_1 застосовна до слова $n(M_1)$, тобто до коду свого власного номера, називається самозастосовною. Передбачається, що машина Тьюринга універсальна, вона читає код свого номера (програми) зі стрічки, розшифровує його й відповідно до нього виконує необхідні дії, також записані на стрічці залежно від початкової конфігурації (даних).

Машина, яка незастосовна до слова $n(M)$, називається несамозастосовною.

Проблема самозастосовності (уперше ця проблема була розглянута вітчизняним математиком О.Б. Лупановим) полягає у тому, щоб за заданою програмою Р для абстрактної машини неможливо було довідатися, чи застосовна вона до свого власного запису P((P)), де (P) - запис програми або підпрограми n(P).

Наприклад, програма машини, що замінює символи 1 на 0, а 0 на 1, застосовна до будь-якого слова, зокрема й до свого запису, якщо ми закодуємо записи програм у двійковому коді, що цілком можливо. Тому вона самозастосовна, а програма В

B: 1) ?{λ,2};

2) HLT,

застосовна тільки до порожнього слова, тобто вона не може бути самозастосовною.

У машині *B*, якщо головка в початковий момент обдивляється чарунку, у якій записаний не пробіл $\wedge$, відбудеться безрезультатна зупинка.

Задача полягає у можливості знаходження такого алгоритму, що визначав би для будь-якої програми її самозастосовність.

Теорема 1.3. Проблема самозастосовності алгоритмічно нерозв'язна.

Нерозв'язності є поширеним явищем, і з їхнім існуванням доводиться рахуватися. З теоретичної точки зору нерозв'язність - не невдача, а науковий факт. Знання основних нерозв'язних проблем теорії алгоритмів має бути для фахівця з дискретної математики таким же елементом наукової культури, як для фізика знання про неможливість створення вічного двигуна. Якщо ж важливо мати справу із розв'язною задачею (а для прикладних наук це прагнення є природним), то варто чітко уявляти собі дві обставини. По-перше (про це уже говорилося під час обговорення проблеми зупинки), відсутність загального алгоритму, що вирішує дану проблему, не означає, що в кожному окремому випадку цієї проблеми не можна домогтися успіху. Тому, якщо проблема нерозв'язна, треба шукати її розв'язні окремі випадки. По-друге, явище нерозв'язності - це, як правило, результат надмірної абстрактності задачі (або мови, на якій описано об'єкти задачі). Задача в більш загальній постановці має більше шансів виявитися нерозв'язною. Крім понять можливості розв'язності й нерозв'язності принципово важливим є поняття складності алгоритмів.

ПРИКЛАДИ Й ПРАКТИЧНІ ЗАВДАННЯ

1.1. Ефективна можливість розв'язання

Приклад 1. Показати МНР-обчислюваність функції $f(x,y)=x+y$.

Розв'язання

Складемо МНР-програму обчислення $f(x,y)$, спираючись на алгоритм додавання 1 до x рівно y раз і використовуючи регістр R_3 як лічильник додавання 1 до $R_1 = x$ при такій початковій конфігурації:

	R_1	R_2	R_3	R_4	
	x	y	0	0	. . .

Програма зупиняє процес обчислень, коли наступить $R_2 = R_3 = y$, і при цьому в регістрі R_1 накопичується число $x + y$, що й потрібно.

МНР-програма :

$$I_1 = J(3,2,5),$$
$$I_2 = S(1),$$
$$I_3 = S(3),$$
$$I_4 = J(1,1,1).$$

Останов здійснюється командою умовного переходу I_1 до команди I_5, що є відсутнім у програмі. Отже, $f(x,y) = (x + y)$ обчислювана.

Приклад 2. Довести МНР-обчислюваність функції

$$g(x,y) = \begin{cases} 0, \text{ якщо } x \le y, \\ 1, \text{ якщо } x > y. \end{cases}$$

Розв'язання

g(x,y) може бути обчислена за алгоритмом, поданим такою блок-схемою $X' = X; Y' = Y$ (рис. 1.2):

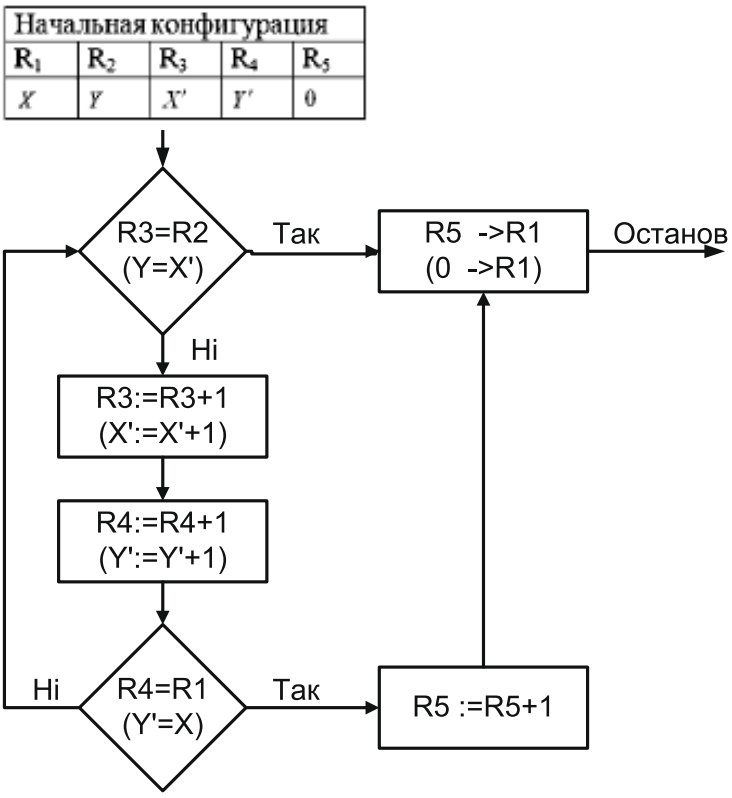


Рис. 1.2. Блок-схема обчислення g(x,y)

МНР-програма має вигляд

$I_1 = J(3,2,7)$;

$I_2 = S(3)$;

$I_3 = S(4)$;

$I_4 = J(4,1,6)$;

$I_5 = J(1,1,1)$;

$I_6 = S(5)$;

$I_7 = T(5,1)$.

Ефективну обчислюваність $g(x,y)$ доведено.

Вправи

1. Покажіть МНР-обчислюваність таких функцій:

а) $f(x) = 10$;

б) $f(x) = \begin{cases} 0, & \text{якщо } x = 0, \\ 1, & \text{якщо } x \neq 0; \end{cases}$

в) $f(x,y) = \begin{cases} 0, & \text{якщо } x = y, \\ 1, & \text{якщо } x \neq y; \end{cases}$

г) $\varphi(x) = \begin{cases} \frac{1}{3}x, & \text{якщо } x \text{ кратне } 3, \\ \text{не визначена} \\ \text{в противному разі;} \end{cases}$

д) $g(x,y) = \begin{cases} 0, & \text{якщо } x \leq y, \\ 1, & \text{якщо } x > y. \end{cases}$

2. Покажіть, що для кожної команди переадресації T(m,n) при будь-якій конфігурації МНР існує програма, що не містить T(m,n).

3. Доведіть можливість розв'язання таких предикатів:

а) $R(x,y) = (x \neq y)$;

б) $Q(x) = (x \neq 0)$;

в) $R_1(x,y) = (x < y)$;

г) $Q_1(x) = (x$ парно).

4. Покажіть, що функція $f(x) = x - 1$ на Z обчислювана.

5. Покажіть, що предикат $Q(x) = (x \geq 0)$ на Z обчислюваний.

1.2. Рекурсивні функції

1. Довести, що будь-яка примітивно-рекурсивна функція усюди визначена.

2. Довести, що з O^1 і I_m^n за допомогою суперпозиції і схем примітивної рекурсії не можна одержати функції $x+1$ і $2x$.

3. Застосовуючи оператор примітивної рекурсії за змінними x_2 і x_3 до функцій $g(x_1)$ і $h(x_1,x_2,x_3)$, одержати функцію $f(x_1,x_2)$. Записати функцію $f(x_1,x_2)$ в аналітичній формі:

а) $g(x_1) = x_1$, $h(x_1,x_2,x_3) = x_1 + x_3$;

б) $g(x_1)=x_2$, $h(x_1,x_2,x_3)=x_1+x_3$;

в) $g(x_1)=2^{x_1}$, $h(x_1,x_2,x_3)=x_3^{x_1}$ (прийняти $0^0=1$);

г) $g(x_1)=1$, $h(x_1,x_2,x_3)=x_3(1+sg|x_1+2-2x_3|)$.

4. Застосувати оператор мінімізації до функції f за змінною x_i. Результуючу функцію зобразити в «аналітичній формі»:

а) $f(x_1)=3$, $i=1$;

б) $f(x_1)=[x_1/2]$, $i=1$;

в) $f(x_1,x_2)=x_1+x_2$, $i=2$;

г) $f(x_1,x_2)=I_1^{(2)}(x_1,x_2)$, $i=2$;

д) $f(x_1,x_2)=x_1-x_2$, $i=1,2$;

е) $f(x_1,x_2)=2^{x_1}+(2x_2+1)$, $i=1,2$.

5. Застосувавши операцію мінімізації до відповідної примітивно-рекурсивної функції, довести, що функція f є частково-рекурсивною:

а) $f(x_1)=x_1/2$;

б) $f(x_1)=2-x_1$;

в) $f(x_1,x_2)=x_1-2x_2$.

1.3. Машина Тьюринга

Приклад 1. Побудувати *Т-машину* з алфавітом $A=\{\wedge,0,1\}$, яка будь-яке скінченне слово з одиниць перетворювала б у слово тієї ж довжини, але з нулями замість одиниць, що стоять на непарних місцях.

Розв'язання

Дія шуканої *Т-машини*, мабуть, полягає у пересуванні по стрічці ліворуч-праворуч і заміні через один символ 1 на 0; зупинитися *Т-машина* повинна при першому обдивлянні порожньої чарунки, тобто символу $\wedge$. Ці дії забезпечує Q_T-програма $g_1 1 0 g_2$, $g_2 0 \Pi g_2$, $g_2 1 \Pi g_1$ з початковою конфігурацією на стрічці вигляду

...	∧	∧	1	1	1	...	1	1	∧	∧	...
			g1								

Заключна конфігурація на стрічці має такий вигляд:

...	∧	∧	0	1	0	...	1	0	∧	∧	...
									g2		

якщо вихідне слово містить парне число "1" або конфігурацію вигляду

...	∧	∧	0	1	0	...	0	1	∧	∧	...
									g1		

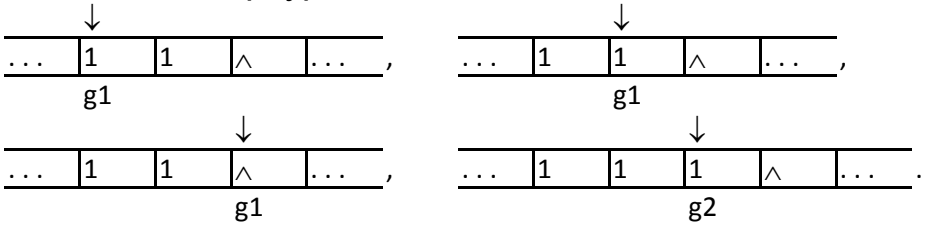
при непарному числі "1" у вихідному слові. *Т-машина* зупиняється тому, що, обдивляючись порожній осередок у Q_T, не знаходить команду, яка визначає подальші дії *Т-машини*.

При обчисленні числових функцій (тобто визначених на N і таких, що набувають значення з N) на машинах Тьюринга користуються спеціальним кодуванням чисел. Наприклад, натуральне число m будемо задавати набором з $m+1$ одиниць і позначати через 1^{m+1}. Тоді нуль кодується 1, одиниця - 11, двійка - 111 і т.д.

Приклад 2. Побудувати *T_1-машину* із зовнішнім алфавітом $A=\{\wedge, 1\}$, що обчислює функцію $S(x) = x+1$.

Розв'язання

Очевидні два стани *T_1-машини* - g_1 і g_2, а також програма $Q_T : g_1 1 П g_1, g_1 1 g_2$. Робота *$T_1$-машини* при обчисленні, наприклад *S(1)*, складається з таких конфігурацій:



Вправи

1. Побудувати машину Тьюринга, що перетворює слово α в слово β в алфавіті $\{0,1\}$:

а) $\alpha = 1^n$, $\beta = 1^n 0 1^n$;

б) $\alpha = 0^n 1^n$, $\beta = (01)^n$;

в) $\alpha = 1^n$, $\beta = 1^{2n-1}$;

г) $\alpha = 1^n 0^n 1^n$, $\beta = 1^m 0 1^n$.

2. Побудувати машину Тьюринга, яка по будь-якому вхідному ланцюжку вигляду $0^n 1^m$ визначає, чи правильна рівність $n = m$.

3. Побудувати машину Тьюринга, яка кожне слово $x_1 x_2 ... x_{n-1} x_n$ в алфавіті $\{a,b\}$ перетворить у слово $x_n x_{n-1} ... x_2 x_1$.

4. Побудувати машину Тьюринга, що обчислює предикат симетрії *S(a)*: для будь-якого слова $\alpha = x_1 x_2 ... x_{n-1} x_n$ в алфавіті $\{a,b\}$ $S(a)=1$, якщо $x_i = x_{n-i+1}$ для всіх $i = 1,2,...,n$ і $S(a) = 0$ – у противному разі.

5. Побудувати машину Тьюринга, що є застосовною до слів вигляду $1^{3n} (n \geq 1)$ і не застосовною до слів вигляду $1^{3n+m} (n \geq 1; m = 1,2)$.

6. Побудувати машину Тьюринга, що обчислює числову функцію

$$I_3^{(4)}(x_1, x_2, x_3, x_4) = x_3.$$

7. Побудувати машину Тьюринга, що обчислює числову функцію

$$sg(x) = \begin{cases} x - y, & \text{якщо } x > y; \\ 1, & \text{у противному разі.} \end{cases}$$

8. Побудувати машину Тьюринга, що обчислює числову функцію

$$sg(x)=\begin{cases}0, & \text{якщо } x=0;\\ 1, & \text{у противному разі.}\end{cases}$$

9. Побудувати машину Тьюринга, що обчислює числову функцію

$$\overline{sg}(x)=\begin{cases}1, & \text{якщо } x=0;\\ 0, & \text{у противному разі.}\end{cases}$$

10. Побудувати машину Тьюринга, що обчислює числову функцію

$$f(x,y)=min\{x,y\}.$$

11. Побудувати машину Тьюринга, що обчислює числову функцію

$$f(x,y)=2x+1.$$

12. Побудувати машину Тьюринга, що обчислює числову функцію $f(x,y)=[x/2]=m$, якщо $x=2m$ або $x=2m+1,\ m\geq 0$.

13. Чи можуть бути застосовні машини Тьюринга T_1 і T_2, які задані командами

$$T_1: q_1 1\to 1\Pi q_1, q_1 0\to 0Hq_0, q_1\Lambda\to\Lambda\Pi q_1;$$

$$T_2: q_1 1\to\Lambda\Pi q_1, q_1 0\to 1Лq_2, q_1\Lambda\to 1Hq_0,$$

$$q_2 1\to\Lambda\Pi q_1, q_2 0\to 0Лq_2, q_2\Lambda\to 0\Pi q_1,$$

до слів а-г?

а) 1111111;

б) 0110111;

в) 111101;

г) 001101.

1.4. Обчислюваність за Марковим

1. Побудувати граф реалізації алгоритму в алфавіті $A=\{+,\perp,?,Q\}$, заданим підстановками $?\perp'\to'\perp','+\perp'\to'?',''?Q'\to'+'$:

а) визначити, до якого виду нормальних алгоритмів він належить;

б) розглянути на ньому приклади дедуктивних ланцюжків, задаючи вихідне слово довжиною не менше трьох символів.

2. Задати нормальний алгоритм Маркова, що реалізує вирахування $A-B$, де значеннями A і B є натуральні числа, подані рядками, що складаються із символів 1 (наприклад, для $A=4,\ B=3,\ A-B=1$ слово ‘1111—111’ має бути перероблене алгоритмом у слово ‘Г).

Перевірити роботу алгоритму для таких випадків:

а) $A=6, B=2$;

б) $A=3, B=5$.

Нормальний алгоритм Маркова, що реалізує операцію множення, задано алфавітом $A=\{1,*,T,\Phi\}$ і послідовністю підстановок:

$$*11\to T*1; *1\to T; 1T\to T1\Phi; \Phi T\to T\Phi;$$

$$\Phi 1\to 1\Phi; T1\to T; T\Phi\to\Phi; \Phi\to 1; 1\to 1.$$

Побудувати дедуктивний ланцюжок від слова '111*1111' до слова '11111111111 Г.

Задати нормальний алгоритм Маркова, що реалізує операцію множення чисел, поданих у вигляді послідовності одиниць (алгоритм повинен бути відмінний від базового алгоритму).

3. Побудувати дедуктивний ланцюжок для одного із вхідних слів.

Задати нормальний алгоритм, що реалізує операцію порівняння призначеної частини двох чисел, заданих у вигляді

а) $\vee\vee\vee 111 * \vee\vee\vee 111$ або $\vee\vee 11 * \vee\vee\vee\vee 1111;$

б) $\vee\vee\vee * 111 - \vee\vee * 111$ або $\vee\vee - 111 * \vee\vee\vee - 11;$

в) $111 - 11 * 11 - 1111.$

У пп. *а і б* призначена частина числа подана символами $\vee$, а в п. в - тими ж символами, що й саме число, тобто одиницями.

Побудувати відповідні дедуктивні ланцюжки по кожному алгоритму.

КОНТРОЛЬНІ ЗАПИТАННЯ Й ЗАВДАННЯ ДЛЯ САМОКОНТРОЛЮ

1) Коли зародилася теорія алгоритмів?
2) Які проблеми привели до появи теорії алгоритмів?
3) Що є предметом вивчення теорії алгоритмів?
4) Із чиїм ім'ям пов'язано поняття «алгоритм»?
5) Який алгоритм називають жадібним?
6) Для чого необхідна формалізація поняття алгоритму?
7) Які існують види моделей уточнення поняття «алгоритм»?
8) Назвіть авторів основних типів алгоритмічних моделей.
9) Що називають алфавітом, словом, композицією слів, підсловом слова, довжиною слова?
10) Якими методами користуються при розробленні алгоритмів?
11) Які слова називають рівними?
12) Виходячи з чого можна обмежитися тільки числовими функціями при вивченні обчислювальних функцій?
13) Що називають функцією, областю визначення, областю значень?
14) Які функції називаються частковими, усюди визначеними?
15) Що таке функціональний алфавіт?
16) Що називають термом?
17) Які функції називають обчислюваними?
18) Які множини називають розв'язними? Які властивості вони мають?
19) Які властивості мають графіки обчислюваних функцій?
20) Які властивості образу й прообразу перераховної множини у понятті обчислюваних функцій?
21) Концепції числового ряду Фібоначчі.
22) Яка ідея побудови класу частково-рекурсивних функцій?
23) Яка суть теореми Кліні?

24) Які найпростіші функції входять до базового набору при побудові класу частково-рекурсивних функцій?

25) Теза Тьюринга.

26) Яка суть машини Тьюринга?

27) Для чого призначено нормальні алгоритми?

ТЕМИ ДЛЯ САМОСТІЙНОЇ РОБОТИ

1) Алгебра розв'язних множин.
2) Алгебра перераховних множин.
3) Програмування для РАМ.
4) Функції, що обчислюються на РАМ.
5) Приклади алгоритмічно нерозв'язних проблем.
6) Моделі обчислень, відмінні від РАМ.
7) Доказ рівносильності будь-яких двох різних моделей обчислень.
8) Приклади задач, що належать до класів P і NP.
9) Приклади NP-повних задач.
10) Обчислення з оракулом.
11) Перераховні множини й обчислювані функції.
12) Зв'язок між перераховними й розв'язними множинами.
13) Дерева висновку.

2. ОСНОВИ ТЕОРІЇ ФОРМАЛЬНИХ ГРАМАТИК

2.1. ПОНЯТТЯ ФОРМАЛЬНОЇ ГРАМАТИКИ. ІЄРАРХІЯ ХОМСЬКОГО

У теорії мов розглядаються принципи й особливості побудови різних мов. До початку XX століття існували тільки природні (розмовні) мови. При цьому під мовою розумівся засіб спілкування між людьми. З розвитком лінгвістики було встановлено, що засоби спілкування властиві не тільки людині. У цей час під мовою розуміється будь-який засіб спілкування.

Мова містить такі складові частини:

- знакову систему (множину припустимих послідовностей знаків);
- множину змістів цієї системи;
- відповідність між послідовностями знаків і змістами.

Знаками мови можуть бути:

- символи (букви) деякого алфавіту (письмова форма мови);
- звуки (усна форма мови);
- жести, колір, заходи тощо.

Найбільш розвиненими є знакові системи на основі символів. Символ є найпростішим елементом знакової системи. Із символів будуються більш складні конструкції. При аналізі розмовних мов ієрархія конструкцій мови має такий вигляд:

Буква Þ слово Þ пропозиція Þ текст.
(знак, символ) (фраза)

У теорії формальних мов використовується формальний підхід, при якому набір конструкцій мови має вигляд

Символ Þ рядок Þ текст.
(знак, буква) (ланцюжок, пропозиція)

У будь-якій мові можна виділити правильні (припустимі) і неправильні конструкції. Правила побудови правильних текстів становлять ***синтаксис*** мови, а опис відповідності між змістами й текстами - ***семантику*** мови.

Семантика мови залежить від походження й характеру мови, тобто від характеру об'єктів, що описуються мовою. Синтаксис мови менше залежить від характеру мови. Тому при вивченні синтаксису можна використовувати формальний підхід.

Суть формального підходу полягає у тому, що мова розглядається як множина формальних об'єктів, побудованих за певними правилами. Формальними об'єктами є послідовності символів. Під час побудови таких послідовностей їхній зміст не враховується. Поява й розвиток формального підходу пов'язані з необхідністю розв’язання задач такого типу:

- машинний переклад з однієї природної мови на іншу;
- розроблення трансляторів;
- розпізнавання образів.

Залежно від походження й ступеня універсальності мови можна розділити на типи, показані на рис. 2.1.

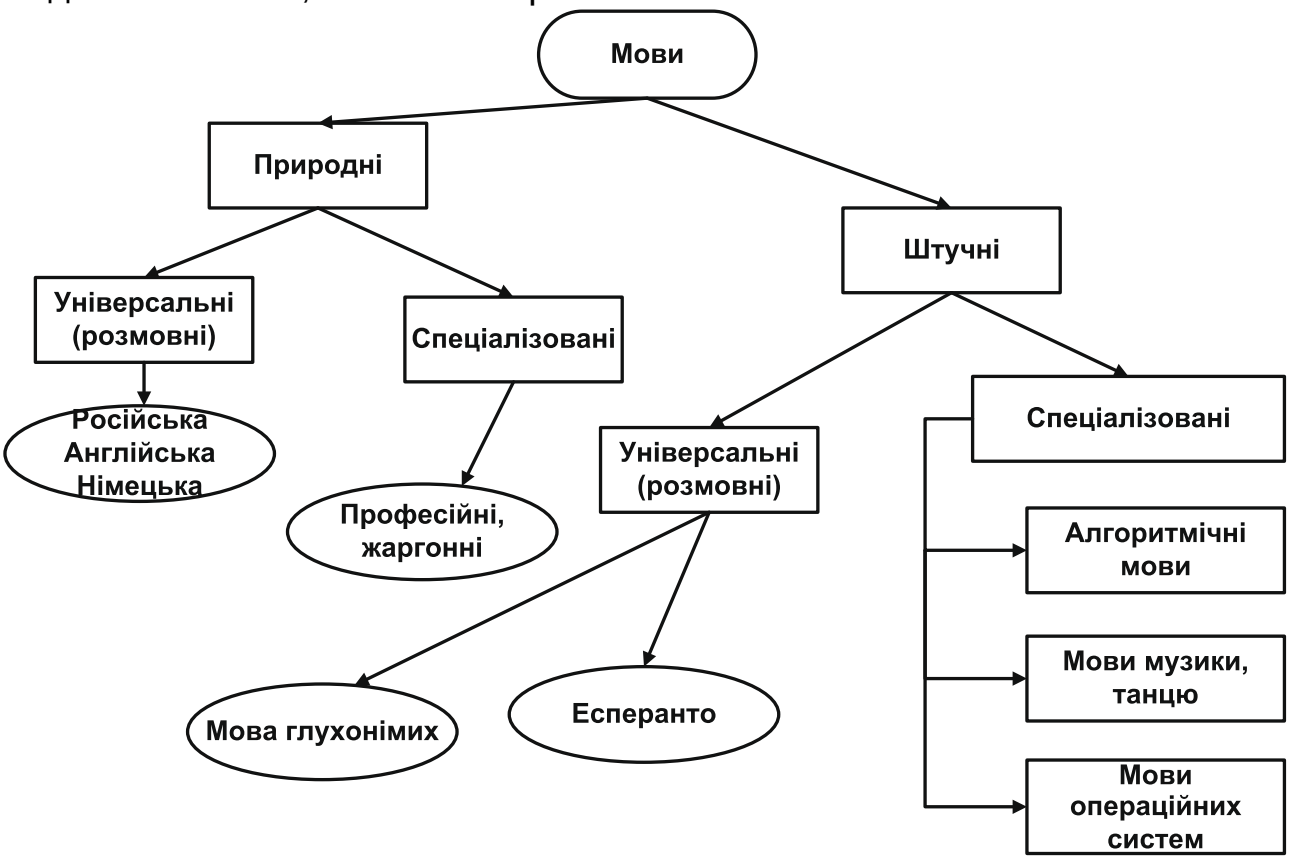


Рис. 2.1. Типи мови залежно від походження і ступеня

Природні мови виникають і розвиваються поступово із розвитком суспільства протягом тривалого часу.

Штучні мови розробляються спеціально для певної галузі застосування за відносно короткий період часу.

Універсальні мови використовуються для спілкування людей у повсякденному житті.

Спеціалізовані мови є засобом спілкування досить вузького кола людей при обміні інформацією у деякій спеціальній області знань. Прикладами спеціалізованих мов можуть бути різні професійні жаргони (мова користувачів ЕОМ), мова алгебри, мова алгебри логіки тощо.

Формальні системи - це системи операцій над об'єктами, під якими розуміється послідовність символів. Передбачається, що між символами не існує ніяких зв'язків і відносин, крім тих, які явно описані засобами самої формальної системи.

Задача. Упорядкувати об'єкти 53, 109, 3.

Швидше за все їх необхідно розставити у порядку 3,53,109 , тобто цій задачі буде дана звичайна арифметична інтерпретація: послідовність цифр розглядається як зображення чисел у десятковій системі; упорядкування цих послідовностей є розташування зображуваних ними

чисел за зростаннями, а правила порівняння таких зображень чисел відомі настільки добре, що про них ніхто й не замислюється.

У дійсності таке тлумачення задачі не випливає з її тексту. Його можна розуміти як задачу лексикографічного впорядкування (і тоді результат буде 109,3,53), як задачу розподілу бігунів із зазначеними номерами по доріжках (розв’язання якої пов'язане із процедурою розподілу й свідомо не пов'язане із числовою інтерпретацією об'єктів) тощо.

Можливість неоднозначного добування задач із зазначеного тексту означає, що цей текст не містить формального визначення задачі. Для такого визначення необхідно чітко описати клас об'єктів, для яких задача розв’язується, і явно ввести для них поняття упорядкування, описавши його як систему локальних операцій над символами, з яких ці об'єкти складаються.

Історично поняття формальної системи виникло в рамках основ математики при дослідженні будови аксіоматичних теорій і методів доказу у таких теоріях.

Усяка точна теорія визначається, по-перше, мовою, тобто певною множиною висловлювань, що мають зміст з точки зору цієї теорії, і, по-друге, сукупністю теорем - підмножиною мови, що складається з висловлювань, істинних у даній теорії.

Однією з фундаментальних ідей в основах математики є ідея формалізації теорій, тобто послідовного проведення аксіоматичного методу побудови теорій. При цьому не допускається користуватися будь-якими припущеннями про об'єкти теорії, крім тих, які явно виражені у вигляді аксіом; аксіоми розглядаються як формальні послідовності символів, а методи доказів - як методи одержання одних виразів із інших за допомогою операцій над символами.

Такий підхід гарантує чіткість вихідних тверджень і однозначність висновків, однак створюється враження, що осмисленість і істинність у формалізованій теорії не відіграють ніякої ролі. Однак у дійсності аксіоми й правила висновку прагнуть вибирати так, щоб побудованою за їх допомогою формальною теорією можна було додати змістовний смисл.

Більш конкретно формальну теорію будують у такий спосіб:

1. Визначають множину формул або правильно побудованих виразів, що утворять мову теорії. Ця множина задається конструктивними засобами (як правило, індуктивним визначенням) і, отже, вона є перераховна, а в деяких випадках і розв’язна.

2. Виділяють підмножину формул, так званих аксіом теорії. Ця підмножина може бути й нескінченною, але у всякому разі вона має бути розв'язною.

3. Задають правила висновку теорії. Правило висновку - це обчислюване відношення на множині формул. Формули називаються посилками правила, а саме - його наслідком або висновком.

Висновком формули B із формули $A_1, A_2, ..., A_n$ називається послідовність формул $F_1, F_2, ..., F_n$ така, що $F_m = B$, а будь-яка F_i є або аксіома, або одна з вихідних формул $A_1, A_2, ..., A_n$, або безпосередньо виведена з F_1, F_{i-1} за одним із правил висновку.

B виведена з $A_1, A_2, ..., A_n$, якщо існує висновок B з $A_1, A_2, ..., A_n$. Цей факт позначається $A_1, A_2, ..., A_n . B$ у $A_1, A_2, ..., A_n$ називаються гіпотезами або посилками висновку.

Доведенням формули B у теорії T називається висновок Y з порожньої множини формул, тобто висновок, у якому як вихідні формули використовуються тільки аксіоми. Формула B, для якої існує доведення, називається формулою, довідною у теорії T, або теоремою теорії T.

Факт довідності формули B позначається $\mapsto B$. Очевидно, що приєднання формул до гіпотез не порушує вивідність. Тому, якщо B - довідна, то A у B - B довідна із деякою формулою A.

При вивченні формальних теорій мають справу з двома типами висловлювань:

1) з висловлюваннями самої теорії, тобто теоремами, які розглядаються як чисто формальні об'єкти, що були визначені раніше;

2) висловлюванями про теорію (про властивості теорем, доказів тощо), які формулюються мовою, зовнішньою стосовно теорії, і називаються метатеоремами.

Термінальні символи - це символи алфавіту T. Нетермінальні символи утворюють множину символів N, що не входять у T, і які використовуються на проміжних кроках породжувальних процесів.

Початковим символом називається нетермінальний символ, з якого виводяться усі рядки мови.

Формальна граматика або просто граматика в теорії формальних мов - спосіб опису формальної мови, тобто виділення деякої підмножини з множини всіх слів деякого скінченного алфавіту. Розрізняють породжувальні і розпізнавальні (або аналітичні) граматики - перші задають правила, за допомогою яких можна побудувати будь-яке слово мови, а другі дозволяють за даним словом визначити, входить воно в мову чи ні [6].

Термінал (термінальний символ) - об'єкт, безпосередньо присутній у словах мови, що відповідає граматиці, і які мають конкретне, незмінне значення (узагальнення поняття «букви»). У формальних мовах, що використовуються на комп'ютері як термінали звичайно беруть усі або частину стандартних символів ASCII - латинські букви, цифри, спеціальні символи.

Нетермінал (нетермінальний символ) - об'єкт, що позначає будь-яку сутність мови (наприклад, формула, арифметичний вираз, команда) і не має конкретного символьного значення.

Сам по собі породжувальний процес полягає у застосуванні на кожному кроці одного з правил перетворень або продукції. Цей процес перетворить заданий рядок у новий рядок; процес закінчується або коли жодна із продукцій не може бути застосована, або коли рядок складається з одних термінальних символів.

Формальна граматика G є четвірка $G=(N,T,E,P)$, де N - множина нетермінальних символів; T - множина термінальних символів; E - початковий символ; P - множина продукцій і $N \cap T \neq 0$ $\alpha \to \beta$, ‹ŜCE α, $\beta \in (\mathrm{N} \cup \mathrm{T})$, $\alpha \neq \lambda$.

Кожний новий рядок у процесі висновку повинен виходити із уже виведеного рядка із застосуванням продукції.

Пропозиція - це рядок, що складається тільки з термінальних символів, виведений з початкового символу.

Мова L, зумовлена граматикою G, є множиною пропозицій, виведених у G у L: $L(G)=\left\{\varpi \in T^* / E \overset{*}{\Rightarrow} \varpi\right\}$.

Ієрархія граматик Хомського.

Американський вчений Ноам Хомський запропонував класифікацію породжувальних граматик на чотири типи залежно від вигляду їхніх правил.

Тип 0. Довільні граматики. На вигляд їхніх правил не накладаються будь-які обмеження. Правила мають вигляд

$$\alpha \to \beta, \tag{2.1}$$

де α і β - ланцюжки терміналів і нетерміналів. Ланцюжок α не повинен бути порожнім.

Тип 1. Контекстно-залежні граматики. Правила таких граматик мають вигляд

$$\alpha A \beta \to \alpha \gamma \beta, \tag{2.2}$$

де α, γ, β - ланцюжки термінальних і нетермінальних символів; A - нетермінальний символ. Такий вигляд правил означає, що нетермінал A може бути замінений ланцюжком γ тільки в контексті, утвореному ланцюжками α і β.

Тип 2. Контекстно-вільні граматики. Їхні правила мають вигляд

$$A \to \gamma, \tag{2.3}$$

де A - нетермінал; γ - ланцюжок терміналів і нетерміналів. Характерна риса - у лівій частині правил є завжди один нетермінал.

Тип 3. Автоматні граматики. Усі правила автоматних граматик мають одну з трьох форм:

$$A \to aB,\ A \to a,\ A \to \varepsilon, \qquad (2.4)$$

де $A,\ B$ - нетермінали; a - термінал; ε - порожній ланцюжок.

Автоматні граматики також називаються регулярними.

Як видно з визначень, кожна наступна граматика є поодиноким випадком попередньої.

Мови, породжувані граматиками типу 0 - 3, називаються відповідно мовами без обмежень, контекстно-залежними, контекстно-незалежними і автоматними (регулярними). Прийнято вважати, що, наприклад, контекстно-вільною є мова, для якої існує контекстно-вільна, але неавтоматна граматика. Так само визначають і контекстно-залежні мови, і мови без обмежень.

Приклади граматик різних типів. Розглянемо граматику G_6, що породжує мову $L_6 = \{a^n b^n c^n \mid n \geqslant 0\}$:

$$G_6 : S \to aSBc \quad (тип\ 2),$$
$$S \to abc \quad (тип\ 2),$$
$$cB \to Bc \quad (тип\ 0),$$
$$bB \to bb \quad (тип\ 1),$$
$$S \to \varepsilon \quad (тип\ 3).$$

Типом граматики вважають мінімальний з типів її правил. Отже, граматика G_6 має належати до типу 0.

Прикладом контекстно-вільних граматик може служити граматика арифметичних виразів. За їх допомогою задається й синтаксис мов програмування. Для прикладу розглянемо таку граматику:

$$G_7 : S \to a \quad (1),$$
$$S \to Sa \quad (2),$$
$$S \to Sb \quad (3).$$

Це граматика типу 2 (правило 1 - тип 3, правила 2 і 3 - тип 2). Розглянемо мову L_7: ланцюжок a будується за правилом 1. Якщо до правильної пропозиції S праворуч приписати a або b, знову вийде правильна пропозиція. Ланцюжок мови L_7 починається з a, далі йдуть a і b у довільному порядку. Якщо під a мати на увазі букву, a під b - цифру, то G_7 можна вважати граматикою ідентифікаторів.

Сконструюємо автоматну граматику, що породжує мову ідентифікаторів.

Граматика G_7 має при цьому такий вигляд:

$$G_8 : S \to a. \qquad (2.5)$$

Позначимо через B частину ідентифікатора, яка може виникати за першою буквою. Тоді можна записати правило

$$S \to aB. \qquad (2.6)$$

Запишемо формулу для B. «Хвіст» може бути буквою або цифрою:

$$B \to a; \qquad (2.7)$$
$$B \to b. \qquad (2.8)$$

Записавши «хвіст» за буквою або цифрою, знову одержимо правильний хвіст:

$$B \to aB; \qquad (2.9)$$
$$B \to bB. \qquad (2.10)$$

Граматика G_8 еквівалентна G_7.

2.2. КЛАСИ ФОРМАЛЬНИХ ГРАМАТИК

Класом граматики прийнято вважати мінімальний клас, у який вона попадає. Наприклад, граматика G вважається контекстно-вільною, якщо вона є граматикою класу 2, але не є граматикою класу 3. Це не перешкоджає тому, що G може бути еквівалентною деякій регулярній граматиці.

Зазначимо, що, не дивлячись на відносну стійкість наведеної у підрозд. 2.1 класифікації граматик Хомського, у деяких авторів можна зустріти різночитання. Наведемо короткий огляд варіантів визначення цих класів.

1. У визначенні контекстно-залежної граматики (клас 1) іноді немає необхідності, щоб $\gamma \in (NUT)^+$, тобто щоб ланцюжок γ, що замінює нетермінал A, був непорожнім. Будь-яка така граматика або така, що еквівалентна якійсь контекстно-залежній у колишньому змісті, або може бути перероблена в еквівалентну з тими ж умовами, що у вихідному визначенні, але з одним-єдиним ε-правилом $S \to \varepsilon$.

2. Визначення контекстно-залежної граматики таке: це граматики, усі правила яких мають вигляд $a \to \gamma$, де a, γ - будь-які слова з терміналів і нетерміналів, але в a вони обов'язково є нетермінальними, а довжина γ не менше, ніж довжина a. Можна довести, що всі такі граматики еквівалентні граматикам класу 1, і навпаки.

3. Граматики класу 3 називають праворегулярними. Разом з ними визначають аналогічне поняття ліворегулярної граматики - коли всі правила мають вигляд

$$A \to Ba;$$
$$A \to a; \qquad (2.11)$$
$$A \to \varepsilon.$$

Можна довести, що кожна ліворегулярна граматика еквівалентна ліворегулярній, і навпаки. Тому ці граматики називають просто

регулярними.

4. Відомо також поняття так званої лінійної граматики. Правила висновку лінійної граматики мають один з таких виглядів:

$$\begin{aligned} A &\to aB; \\ A &\to Ba; \\ A &\to a; \\ A &\to \varepsilon. \end{aligned} \qquad (2.12)$$

На жаль, існують лінійні граматики, які не є еквівалентними ніякий регулярній з грамитик. Такою, наприклад, є граматика G

$$\begin{aligned} S &\to aB; \\ B &\to Sb; \\ B &\to b, \end{aligned} \qquad (2.13)$$

що породжує мову

$$L = \{a^n b^n \mid n > 0\}.$$

Дійсно, легко бачити, що ця мова задається граматикою $H : S \to aSb, S \to ab$, правила якої виводяться з даних продукцій. Позначивши через B усі значення $a^{n-1}b^n (n > 0)$, із правил H легко одержимо правила G. Звідси H і G еквівалентні. Далі буде доведена теорема (про накачування), з якої слідує нерегулярність мови L.

2.3. КОНТЕКСТНО-ВІЛЬНІ ГРАМАТИКИ

Звернемо увагу на те, що в правилах НВ-граматики замінюється тільки один символ, ліва ж частина правила не обов'язково складається тільки з цього символу: $A \to \omega$. У правилах можуть бути присутніми й інші символи - контексту $\varphi A \psi \to \varphi \omega \varphi$. Такі правила означають дозвіл заміняти символ A на ω тільки в контексті φ і ψ. Сам контекст при цій заміні залишається без зміни.

Правила, що використовують контекст, назвемо контекстно-зв’язаними, а правила, що не використовують контексту – контекстно-вільними.

НВ-граматики, що містять тільки контекстно-вільні правила вигляду $A \to \omega$, називаються *контекстно-вільними* (КВ-граматики), або безконтекстними граматиками.

НВ-граматики, що містять контекстно-зв’язані правила, називаються *контекстно-зв’язаними* граматиками.

Мови, що породжуються КВ-граматиками, називаються *КВ-мовами*.

Зазначимо, що зв'язаними контекстом, або вільними, є тільки правила, а не елементи в термінальному ланцюжку.

КВ-граматики являють собою важливий окремий випадок НВ-граматик. Їхня цінність зумовлена такими двома обставинами:

- по-перше, відмова від контексту, тобто вимога, щоб у лівій частині правила був рівно один символ, робить структуру граматик ще більш простою, що полегшує її вивчення;

- по-друге, хоча в природних мовах заміна одних одиниць іншими часто припустима лише в певних контекстах, доцільно досліджувати можливість описувати мови, відволікаючись від зазначеного факту. У природних мовах можливі ситуації, коли явища, які подаються істотно залежними від контексту, можуть описуватися і як незалежні від контексту, тобто в термінах КВ-граматики. При цьому опис може ускладнюватися в інших аспектах. Наприклад, може знадобитися багато нових категорій, правил або того й іншого.

Загалом така заміна робиться так: нехай є клас елементів X у сусідстві з елементами деякого класу Y, елементи X поводяться інакше, ніж у сусідстві з елементами класу *2,* так що мають місце правила

$$\begin{aligned} YX &\to YAB; \\ ZX &\to ZCD \end{aligned} \qquad (2.14)$$

(правила використовують контекст).

Уведемо два нових символи X_1 і X_2. Елемент X у позиції після Y позначимо через X_1, а в позиції після Z - через X_2.

Тоді приходимо до правил, що не використовують контексту

$$\begin{aligned} X_1 &\to AB; \\ X_2 &\to CD. \end{aligned} \qquad (2.15)$$

Не треба, однак, думати, що всяка контекстно-зв'язана НВ-граматика може бути замінена еквівалентною їй КВ-граматикою. Відомо, що існують НВ-мови, що не є КВ-мовами, наприклад, мова, що складається зі всяких ланцюжків вигляду $a^n b^n a^n (aba, aabbaa, ...)$ або з ланцюжків вигляду $a^n b^n c^n$. Від контексту не можна відмовитися, якщо правило має забезпечувати перестановку символів, оскільки перестановка за своєю суттю є багатовимірною операцією. Отже, КВ-граматика не може породжувати мову, що містить ланцюжки, які не можуть бути побудовані без застосування перестановок.

Майже всі наявні приклади НВ-мов, що не є КВ-мовами, носять абстрактний характер і не мають інтерпретацій у природних мовах.

Дотепер ми займалися введенням все нових і нових обмежень на класи розглянутих граматик. Спочатку ми зажадали, щоб число символів у правій частині правил було не менше, ніж у лівій, і одержали неукорочувані граматики. Потім зажадали, щоб заміні піддавався тільки один символ, і одержали НВ-граматики. Нарешті, ми зажадали, щоб у лівій частині правила взагалі був тільки один символ, і одержали КВ-граматики.

Ясно, що ніяких подальших обмежень на ліві частини правил накласти вже не можна. Тому, якщо ми хочемо виділити ще більш вузькі класи граматик, доведеться накладати обмеження на праві частини.

Почнемо із числа символів у правій частині. Залежно від числа символів у правій частині правил КВ-граматики можна розділити на *бінарні* й *небінарні.*

КВ-граматики будемо називати *бінарними,* якщо права частина будь-якого правила містить не більше двох символів. Наприклад, правила вигляду $A \to BC$, або $A \to bB$, $A \to B$, де $A,B,C \in V_H, b \in V_T$.

КВ-граматики будемо називати *небінарними,* якщо права частина будь-якого правила містить більше двох символів. Наприклад, граматика з правилами вигляду $A \to Aab$, $B \to ABC$, $A \to \omega$, де $A,B,C \in V_H, a,b \in V_T \omega$, не є порожній ланцюжок у цій граматиці, що містить більше двох символів.

Бінарні КВ-граматики мають ту особливість, що у відповідних їм деревах структури складових S-маркерах з кожної вершини виходить не більше двох гілок. Це значить, що будь-яка складна складова завжди має рівно дві безпосередньо вкладених у неї складові.

Лінійні граматики - такі КВ-граматики, праві частини правил яких містять не більше, ніж по одному входженню нетермінального символу. Таким чином, для бінарних КВ-граматик це є правила вигляду

$$A \to aB, \tag{2.16}$$

де $B, A \in V_H, a \in V_T$, а для небінарних КВ-граматик - правила вигляду

$$\begin{aligned} &A \to aBab, A \to acB; \\ &A,B \in V_H, a,b,c \in V_T. \end{aligned} \tag{2.17}$$

Мова, що породжується лінійними граматиками, називається лінійною мовою. КВ-граматики, праві частини правил яких містять більше одного нетермінального символу, будемо називати *нелінійними граматиками.*

КВ-граматика називається металінійною, якщо праві частини її правил не містять мети граматики, і всі правила, ліві частини яких відмінні від мети, мають такий же вигляд, як правило лінійної граматики.

Прикладом металінійної граматики може служити така граматика:

$$G = (\{a,b,c\},\{S,T\},\{S \to TT, T \to aTa, T \to bTb, T \to c\},S). \tag{2.18}$$

Мова називається металінійною, якщо існує породжувальна його металінійна граматика.

Накладаючи обмеження на склад символів правої частини правил КВ-граматики, Флойд виділив такі підкласи небінарних, нелінійних граматик.

Операційні граматики - праві частини правил, які не можуть містити двох нетермінальних символів, що знаходяться поряд. Прикладами правил такого вигляду можуть служити

$$A \to BbC;$$
$$A \to BabC, \qquad (2.19)$$

де $A,B,C \in V_H, a,b \in V_T$.

Попередні граматики - праві частини правил, які можуть містити два термінальні символи, що знаходяться поряд. При цьому є можливість указати, який з цих термінальних символів виникає у словотворенні першим, що має більший пріоритет.

Прикладами правил такої граматики є

$$A \to BaaB,$$

де $A,B \in V_H, a \in V_T$. Існують й інші підкласи. Підкласом лінійних КВ-граматик є однобічні лінійні граматики.

Однобічні лінійні граматики - такі граматики, праві частини правил яких містять термінальні символи, але тільки з одного боку від нетермінального символу.

Однобічні лінійні граматики підрозділяються на лівобічні - із правилами вигляду $A \to xB$ і $A \to x$ і правобічні - із правилами вигляду $A \to Bx$ і $A \to x$. В обох випадках A,B - нетермінальні символи, а x - непорожній ланцюжок термінальних символів.

Однобічні лінійні граматики, у яких у кожному правилі ланцюжок x складається тільки з одного символу, називаються автоматними граматиками, або А-г*раматиками*, а мови, що породжуються цими граматиками, - автоматними мовами, або А-мовами.

З даних визначень зрозуміло, що кожний наступний клас граматик утримується в попередньому.

Взаємозв'язок розглянутих класів граматик можна подати у вигляді графа, зображеного на рис. 2.2. Таким чином, КВ-граматики являють собою найбільш важливий підклас НВ-граматик. Це пояснюється такими чотирма основними причинами:

1) КВ-граматики є основою визначення майже всіх уживаних мов програмування;

2) усі дії системи синтаксичного аналізу для природних мов основані на КВ-граматиках;

3) це єдиний тип граматики, теорія якої вивчена й практично перевірена;

4) усі трансформаційні граматики побудовані на КС-граматиках;

5) усі розглянуті типи граматики породжують чотири типи мов: НС-мову, КВ-мову, лінійну мову, А-мову, не враховуючи мови, що породжені найзагальнішим типом граматики з необмеженими правилами висновку, тобто граматикою типу 0.

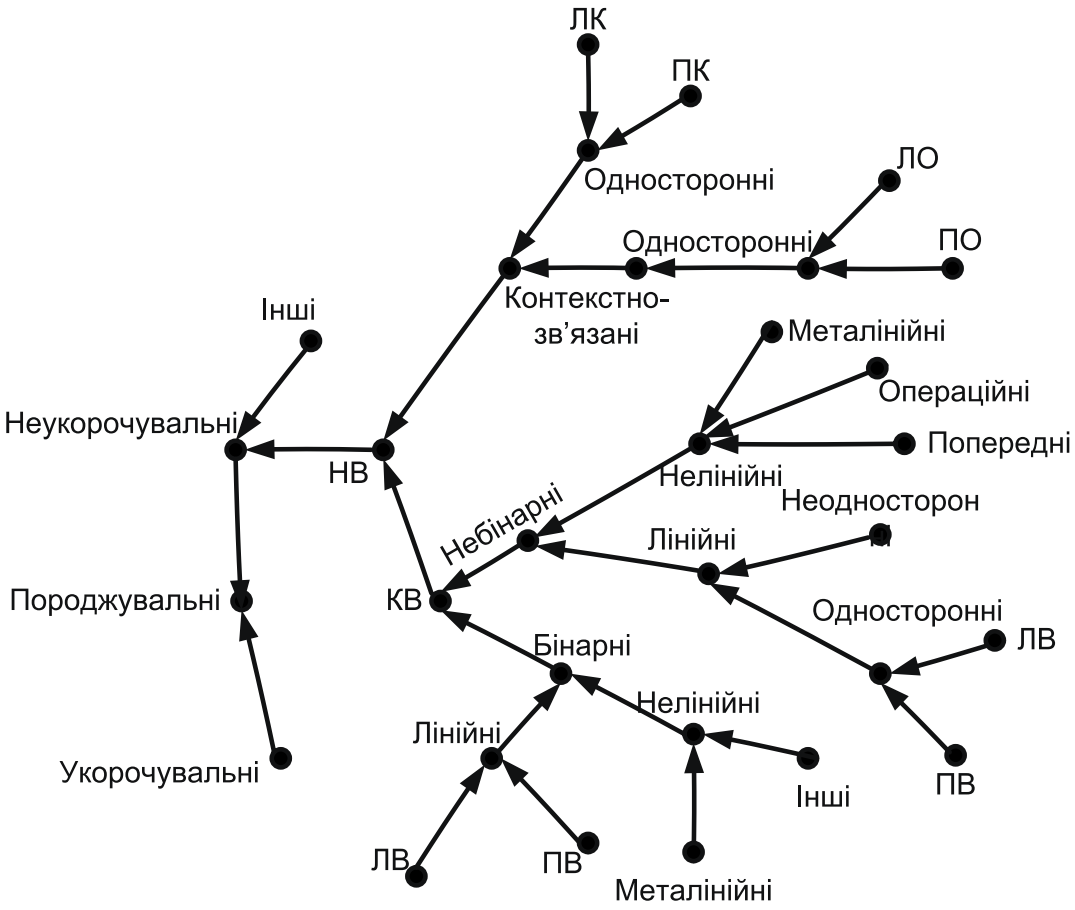


Рис. 2.2. Взаємозв'язок різних класів граматик

Взаємозв'язок між мовами, що породжується розглянутими типами граматик, буде таким:

$$L(О) \supset L(НС) \supset L(КС) \supset L(Л) \supset L(А).$$

2.4. ОСНОВИ ТЕОРІЇ ФОРМАЛЬНИХ МОВ

2.4.1. Властивості формальних мов

Розглянемо ряд теорем, що характеризують основні властивості мов, породжених чотирма основними типами граматик.

Теорема 2.1 *(Поста).* Будь-яка мова типу *0* є рекурсивно-перераховною (хоча, можливо, і нерекурсивною) множиною ланцюжків. Будь-яка рекурсивно-перераховна множина ланцюжків є мовою типу *0.*

З точки зору цієї теореми теорія мов типу *0* охоплюється загальною теорією рекурсивних функцій і тому звичайно в теорії мов мови типу *0* не розглядаються.

Теорема 2.2 *(Хомського).* Мови типу *1, 2, 3* є рекурсивними множинами ланцюжків, тобто для кожної мови із зазначених чипів існує алгоритм, що дозволяє за заданою граматикою визначеної мови розпізнавати приналежність будь-якого ланцюжка мові. Зворотне твердження неправильне, тобто існують рекурсивні множини, що не є

мовами. Це дало підставу для проведення спеціальних досліджень цих мов.

Теорема 2.3 *(Хомського).* Мови типу *3* є регулярними множинами ланцюжків. Тому їх іноді називають автоматними.

Таким чином, якщо клас мов типу *0* виявився настільки широким, що теорія його збіглася із загальною теорією рекурсивних функцій (теорією алгоритмів), то клас мов типу *3* виявився, навпаки, надмірно вузьким, таким, що збігається з добре вивченим класом «регулярних множин (у теорії скінченних автоматів). Тому «особлива увага при побудові теорії мов приділялася мовам/типу 1/*2.*

Теорема 2.4. Існує мова типу *0, що* не є мовою типу *1.* Ця теорема випливає з теорії рекурсивних функцій.

Теорема 2.5. Існує мова типу *1, що* не є мовою типу *2.*

Прикладом такої мови може служити мова $L\{a^n b^m a^n b^m c^3\}$.

Теорема 2.6. Існує мова типу *2, що* не є мовою типу *3.*

Прикладами таких мов є мови

$$\{a^n b^n\} \subseteq \{xx^T\}.$$

Наведений ряд теорем визначає таке відношення між різними типами мов:

$$\{L \ тип \ 3\} \subseteq \{L \ тип \ 2\} \subseteq \{L \ тип \ 1\} \subseteq \{L \ тип \ 0\}.$$

Мови типів *0 - 3* утворюють систему, граматики яких являють собою систему правил єдиного типу з послідовно зростаючими обмеженнями. Однак вони не вичерпують можливостей побудови граматик такого ж вигляду, але з іншими обмеженнями, що породжують мови, які є відмінними від уже розглянутих.

Існують підкласи мов, для яких можуть бути встановлені цікаві закони й властивості, що не мають місця для класу в цілому.

Так, у граматиках типу *1* розглядається мова, що називається мовою Ларіна, граматика якої має додаткові обмеження: заборонені правила вигляду $\varphi_1 A \varphi_2 \to \varphi_1 B \varphi_2$, тобто правила, що містять заміну одного нетермінального символу іншим. Хоча ця мова становить інтерес для математичної лінгвістики, вона вивчена недостатньо.

З усіх чотирьох типів мов найцікавішими є мови типу *2* - безконтекстні.

Багато в чому це визначається можливістю їхнього використання для дослідження мов програмування. Як відомо, програма цифрової обчислювальної машини може розглядатися як ланцюжок символів у деякому алфавіті. Тоді деяка мова програмування являє собою нескінченну множину таких ланцюжків. Ця мова має граматику й скінченний набір правил, що визначають побудову програм. Крім того, мови типу *2* збігаються з алголоподібними мовами, тобто мовами, які можуть бути задані в нормальній формі Бекуса - загальноприйнятій формалізації мов програмування.

У той же час при дослідженні природних мов використовують так звані категоріальні граматики, які дозволяють відбирати з числа всіх можливих пропозицій правильно побудовані. Клас мов, зумовлених категоріальними граматиками, збігається з класом мов, породжуваних граматиками типу *2.*

Мови типу *2* найбільш близькі до мов регулярних подій (типу *3),* з вичерпною повнотою досліджених у теорії скінченних автоматів. Крім того, мови типу *2* одержали адекватне подання за допомогою математичної моделі автомата з магазинною пам'яттю.

Нарешті, мови цього типу більш доступні для математичного вивчення, ніж, наприклад, мови типу *1* або мова Перуки, і тому вони найбільше використовуються при дослідженні поширених мов програмування, а також у математичній лінгвістиці. Зі сказаного стає очевидним те, чому для мов типу *2* отримано найбільшу кількість результатів і досліджено велику кількість їх різновидів, кожна з яких є окремим випадковим безконтекстної мовои.

Розглянемо ще один підклас безконтекстних мов, що визначається за допомогою додаткових обмежень на систему правил.

Нехай є термінальний словник $V_T = \{a_1,...,an\}$. Позначимо через V'_T розширений словник, у якому кожному термінальному символу a_i, зіставлений символ $a'_i : V'_T = \{a_1, a'_1,..., a_n, a'_n\}$. Мова, що породжується граматикою $G = (V'_i, \{s\}, P, S)$, правила якої мають вигляд

$$P: S \to \lambda,$$

$$S \to Sa_iSa'_iS,$$

називається мовою Дика і позначається буквою Д.

Пропозиції ω цієї мови мають таку властивість: якщо кожній парі $a_i a'_i (i = 1, 2,..., n)$ сусідніх символів, що втримуються у ω, дозволити заміняти порожнім символом $\wedge$, то для кожної пропозиції ω знайдеться така послідовність цих замін, за допомогою якої ω зведеться до порожнього ланцюжка. Інтерес до мов Дика пояснюється тим, що вони очевидним чином пов'язані з дужковими структурами, які є звичайними для природних і штучних мов.

Уявимо собі набір формул деякого математичного обчислення або програму, записану на мові типу АЛГОЛ. Це буде текст, у якому зустрічатимуться знаки, що завжди вживаються тільки попарно: ліві й праві дужки всіх видів (круглі, квадратні, фігурні, ламані) або операторні дужки, що складаються зі слів «початок» і «кінець». Усунемо з тексту всі знаки, крім знаків зазначеного типу. Вийде новий текст, побудований за деякими строгими правилами. Подібні тексти дають уявлення про мови Дика.

2.4.2. Операції над формальними мовами

До мов, як і до всяких множин, можуть бути застосовані різні операції. Перш ніж розглядати операції над мовами, визначимо властивість замкнутості множини. Вважають, що множина замкнута щодо деякої операції, якщо результат застосування її до будь-якого елемента множини або до кожної пари елементів залишається у *цій* множині.

Для мов звичайним чином визначаються операції об'єднання, перетинання й операції доповнення щодо фіксованого словника V_T.

Об'єднанням мов L_1 і L_2 (позначення: $L_1 \cup L_2$) називається множина всіх слів, що належать хоча б одній з мов.

Ця операція являє собою звичайне теоретико-множинне об'єднання; вона комутативна й асоціативна:

$$\begin{aligned} &L_1 \cup L_2 = L_2 \cup L_1, \\ &(L_1 \cup L_2) \cup L_3 = L_1 \cup (L_2 \cup L_3). \end{aligned} \qquad (2.20)$$

За тих самих умов перетинанням двох мов (позначення $L_1 \cup L_2$) називається множина всіх слів, що належать одночасно обом мовам.

Ця операція являє собою звичайне теоретико-множинне перетинання; вона теж комутативна й асоціативна.

Доповненням мови L до V_T називається множина всіх слів, що належать V^*_T, але не належать L.

Сама множина V^*_T є мова, доповненням якої до V^*_T є порожня мова.

Підкреслимо, що мова, яка містить порожнє слово e, не є порожньою.

Операцію доповнення розглянемо на такому прикладі:

$$V_T = \{a, b\},$$

$$L = \{a^m b^n \mid m \geq 1, n \geq 1\},$$

$$V^*_T \setminus L = (L_1 \cup L_2 \cup L_3),$$

де L_1 - множина всіх слів, що починаються з L_2, L_2 - множина всіх слів, що починаються з $a^m b^n a$ і $L_3 = \{a\}^*$, тобто L_3 є множина всіх слів, що складаються тільки з a.

Відношення мов типів *0, 1, 2, 3* до булевих операцій визначають такі чотири теореми, які ми наводимо без доведення.

Теорема 2.7. Клас мов типу *0* замкнутий щодо операцій об'єднання й перетинання. Алгоритмічно нерозв'язна задача визначення того, чи є доповнення мови типу *0* щодо фіксованого словника також мовою типу *0.*

Теорема 2.8. Клас мов типу *1* замкнутий щодо операцій об'єднання й перетинання.

Питання про те, якому класу належить доповнення мов типу *1* щодо фіксованого словника, зараз залишається відкритим.

Теорема 2.9 (Бар-*Хіллела, Перльса і Шаміра).* Клас мов типу *2* замкнутий щодо операцій об'єднання, але не замкнутий щодо операції доповнення щодо словника, що містить не менше двох символів, а також щодо операції перетинання.

Теорема 2.10 (*Кліні*). Клас мов типу *3* замкнутий щодо всіх булевих операцій.

Крім булевих операцій над мовами розглядаються також операції множення або конкатенації, ітерації, транспозиції (дзеркального відображення мови), гомоморфізму та деякі інші.

Добутком (конкатенацією) двох мов (позначення $L_1 \cdot L_2$) називається множина всіх слів, які можна одержати у такий спосіб: береться деяке слово з L_1 і до нього приєднується конкатенацією праворуч деяке слово з L_2, тобто

$$L_1 L_2 = \{ X_1 X_2 \mid X_1 \in L_1, X_2 \in L_2 \}.$$

Ця операція, що називається множенням мов, не збігається з декартовим множенням; вона асоціативна, але не комутативна.

Нехай $V_T = \{a, b, c\}$. Розглянемо мову $\{a\}$, що складається з одного однобуквеного слова a. Тоді добуток

$$L = V_T^*$$

є множиною усіх слів, що починаються з a.

Операція ітерації (операція Кліні). Оскільки операція множення мов асоціативна, ми можемо піднести дану мову 1! до степеня:

$$L^2 = LL, L^3 = (LL)L = L(LL), ...$$

Кліні запропонував розглядати об'єднання

$$L^* = E \cup L \cup L^2 \cup L^3 \cup ... \cup L^n \cup ...$$

усіх послідовних степенів мови L. Це об'єднання позначається L^* і називається ітерацією мови L.

Наприклад, ми можемо розглядати алфавіт

$$V = \{a, b, ...\}$$

як мову, що складається з однобуквених слів. Тоді V^2 - множина всіх двобуквених слів; V^3 - множина всіх трибуквених слів і т.д. Тому $E \cup V \cup V^2 \cup V^3 \cup ...$ є множина всіх слів над V, тобто V^*.

Доведено замкнутість класів регулярних і безконтекстних мов щодо множення й ітерації. Мова L називається регулярною, якщо існує скінченний автомат A, такий, що $L = L(A)$.

Щодо мов 0, 1, 2, 3 має місце така теорема.

Теорема 2.11. Класи мов типів 0, 1, 2, 3 замкнуті щодо операції дзеркального відображення, що визначається у такий спосіб: нехай дана мова $L \subset V^*_T$ через L^T позначається мовою, що складається з обігів усіх слів мови L:

$$L^T = \{ X^T \mid X \in L \}.$$

Ця операція є інвалютивною (тобто збігається зі своєю інверсною операцією):

$$(L^T)^T = L.$$

Крім того, вона пов'язана із множенням мов таким співвідношенням:

$$(LM)^T = M^T L^T.$$

Наприклад, мова $V^*_T (V^*_T)^T$ збігається з V^*_T, оскільки $(V^*_T)^T = V^*_T$ і $E \in V^*_T$.

Уведемо тепер поняття про операцію *гомоморфізму.*

Поставимо у відповідність елементу a словника V_T скінченний словник V_{Ta}. Позначимо через V^*_{Ta} множину всіх ланцюжків зі словника V_{Ta}, а через $\tau(a)$ - будь-який ланцюжок з V^*_{Ta}. Таким чином, функція τ визначена на окремих символах a зі словника V_T. Визначимо тепер функцію τ на реченнях зі словника V_T у такий спосіб: якщо $x = ai_1 ai_2 ... ai_s$, то $\tau(x) = \tau(ai_1)\tau(ai_2)...\tau(ai_s)$.

Так, деяка функція τ відображує підмножину ланцюжків L з V^*_T у деяку підмножину ланцюжків з $(\bigcup\limits_a V_{Ta})^*$ *, тобто $\tau(L) = (\bigcup\limits_a V_{Ta})^*$.

Операція $\tau(L)$ відображення мови L за допомогою функції τ називається *операцією гомоморфізму*.

Теорема 2.12 (*Бар - Хіллела, Перльса і Шаміра*). Класи мов типу 0, 2, 3 замкнуті щодо операції гомоморфізму.

Для мов же типу 1 (контекстних) має місце така теорема.

Теорема 2.13 (*Гінсбурга і Роуза*). Якщо V_T містить хоча б два елементи, то клас контекстних мов не замкнутий щодо операції гомоморфізму.

Проекція мови. Для кожного слова в алфавіті $X \times Y$

$$< x(1)y(1) >< x(2)y(2) > ... < x(t)y(t) >,$$

його проекціями в X і Y називаються відповідно слова

$$x(1)...x(t);$$

$$y(1)...y(t).$$

Інакше кажучи, якщо задано мову L в алфавіті $X \times Y$, то проекцією мови L на X називається мова, що складається точно з проекцій у X слів мови L.

Циліндр мови. Нехай задані алфавіт Y і мова L в алфавіті X. Y-циліндром мови L називається мова L', що складається з усіх слів в алфавіті $X \times Y$, X проекції яких належать L.

Властивості мов щодо розглянутих операцій зведено в табл. 2.1.

Таблиця 2.1

Властивості мов залежно від типу

№ п/п	Операція	Тип мови			
		0	1	2	3
1	Об'єднання	1	1	1	1
2	Перетинання	1	1	0	1
3	Доповнення	0		0	1
4	Транспозиція (дзеркальне відображення)	1	1	1	1
5	Добуток (конкатенація)	1		1	1
6	Ітерація	1		1	1
7	Гомоморфізм	1	0	1	1

У цій таблиці одиниця позначає замкнутість щодо відповідної операції класу мов певного типу - нуль-незамкнутість. Порожня клітинка означає, що питання поки що не вирішене.

2.5. МЕТОДИ АНАЛІЗУ ГРАМАТИК

Методи граматичного розбору можна розбити на два великих класи - висхідні й спадні - відповідно до порядку побудови дерева граматичного розбору. Спадні методи (методи зверху вниз) починають із правила граматики, що визначає кінцеву мету аналізу з кореня дерева граматичного розбору й намагаються його нарощувати, щоб наступні вузли дерева відповідали синтаксису аналізованої пропозиції. Висхідні методи (методи знизу вгору) починають із кінцевих вузлів дерева граматичного розбору й намагаються об'єднати їхньою побудовою усе більше й більше вузлів високого рівня доти, доки не буде досягнутий корінь дерева.

Основна проблема теорії мов полягає у тому, щоб формально аналізувати класи мов з метою розроблення можливих методів як моделювання, так і ефективного оброблення мов машинними засобами.

В основному це зводиться до визначення логічної структури мов, тобто системи правил, що визначають синтаксис граматики. Якщо форма

правил (тобто синтаксична частина граматики) точно встановлена, то можливе проведення такого ряду досліджень з мови:

– установлення зв'язку між видами мов з їхніми структурними деревами й формою синтаксичних правил;

– вивчення структурних властивостей мов, які породжуються деякою формою правил G-граматики;

– визначення відносного багатства або бідності різних форм граматик, що породжують L-мови.

– установлення різного роду проблем можливості розв'язання L-мови щодо заданої G-граматики і класу G-граматик;

– установлення потужності G-граматики, що породжує моделюючу L-мову залежно від контексту застосування останнього;

– оцінювання здатності, що породжує G-граматики, а отже, визначення еквівалентності породжуваної різними граматиками L множини мов;

– оцінювання міри складності породжених G-граматиками пропозицій L-мови;

– установлення можливості зведення G-граматик різної складності до більш простих G-граматик;

– визначення можливих методів побудови розпізнавальних G-граматик для множини L-мов і ряд інших не менш важливих досліджень, пов'язаних з вивченням властивостей класу формальних граматик з метою їхнього практичного використання.

Найширшого розвитку на сьогодні набула проблема синтаксичного аналізу граматик і контролю відповідних їм мов, яка для будь-якого ланцюжка $\omega \in L$ визначає їхню формальну правильність (або неправильність) і виконує синтаксичний розбір.

Проблема синтаксичного аналізу й контролю мови виникла у зв'язку з потребами трансляції - розробленням спеціалізованої програмуючої програми, за допомогою якої машина перекладає уведену в неї програму машинною мовою.

Кожний із трансляторів побудований в основному для деякої конкретної пари мов: одного - вхідного і другого - вихідного. Просто кажучи, такі транслятори працюють за принципом словника: для кожної конкретної комбінації символів, що має зміст у даній вхідній мові, цей транслятор видає певну послідовність символів вихідної мови.

Побудова таких трансляторів являє собою досить трудомістку роботу, оскільки потрібно розглянути всі можливі вирази вхідної мови, що мають зміст, і кожному з них зіставити відповідні вирази вихідною мовою.

Визначення виразів вхідної мови, що мають зміст, здійснюється у результаті синтаксичного аналізу цієї мови. Суть його полягає у тому, що на підставі синтаксичних правил граматики перевіряється граматична правильність заданих пропозиції або слів мови шляхом граматичного

розбору. Таким чином, задачею граматичного розбору є аналіз пропозицій для встановлення їхньої граматичної правильності.

Під граматичним розбором розуміється процес визначення структури пропозиції або слова α, $\alpha \in G$ щодо правил, зумовлених G.

Установлення того факту, що пропозиція або слово є граматично правильним, може бути виконано не одним способом. Граматичний розбір може в деяких випадках виконуватися більш ніж одним способом.

Як приклад мови, для якої граматичний розбір може бути здійсне не одним способом, розглянемо мову, що задається граматикою G із сукупністю правил:

$$HBC \to HBc,$$

$$HBC \to hBC,$$

$$BC \to bc,$$

$$HB \to hb.$$

Розглянемо вхідне слово hbc. Це слово може бути розібрано двома різними способами. Дерева граматичного розбору цього слова показано на рис. 2.2.

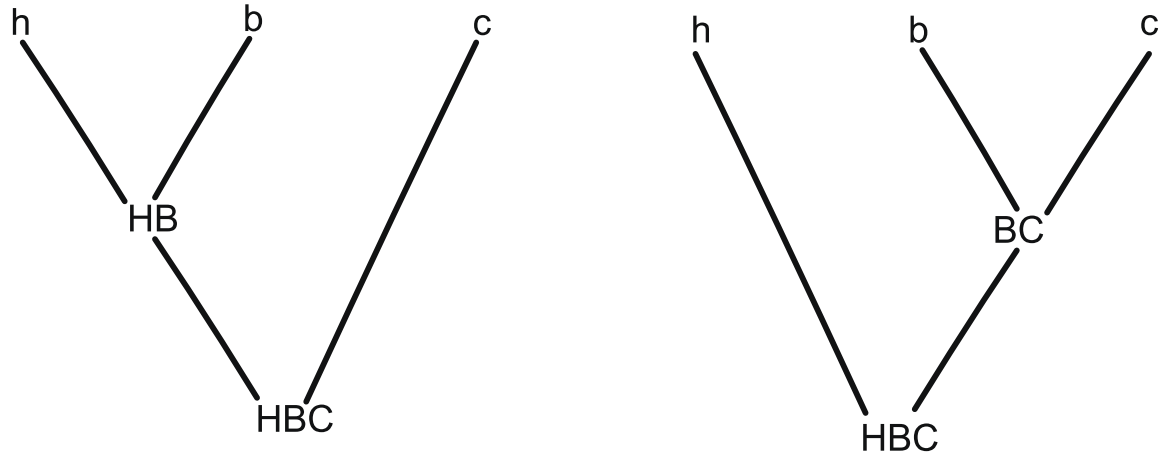


Рис. 2.2. Дерева граматичного розбору слова hbc

У найпростішому вигляді граматичний розбір полягає у тому, щоб, почавши з першого правила, переглядати список правил зверху вниз, доки не буде знайдене застосовне правило, і застосувати його й повторити цей процес стільки разів, скільки потрібно. Цю методику відповідно до наших правил дав би граматичний розбір, показаний на рис. 2.2 праворуч.

Розглянемо ще один метод граматичного розбору, оснований на порядку перегляду слів, що розбираються, або пропозиції. Правила граматики мови мають вигляд

$$A \to BC;$$

$$B \to DE;$$

$$C \to FH.$$

Необхідно виконати граматичний розбір слова $DEFH$.

Розбір даного слова можна проводити *зліва направо* або *справа наліво.* У першому випадку в слові вибирається перша доступна для заміщення сукупність символів DE відповідно до заданих граматик. Замість неї підставляються символи з деякого правила. Після цього отримане слово знову проглядається, починаючи зліва, з метою пошуку сукупності символів для заміщення за правилами граматики.

Для нашого прикладу перегляд зліва направо дає дерево граматичного розбору, показане на рис. 2.3, а.

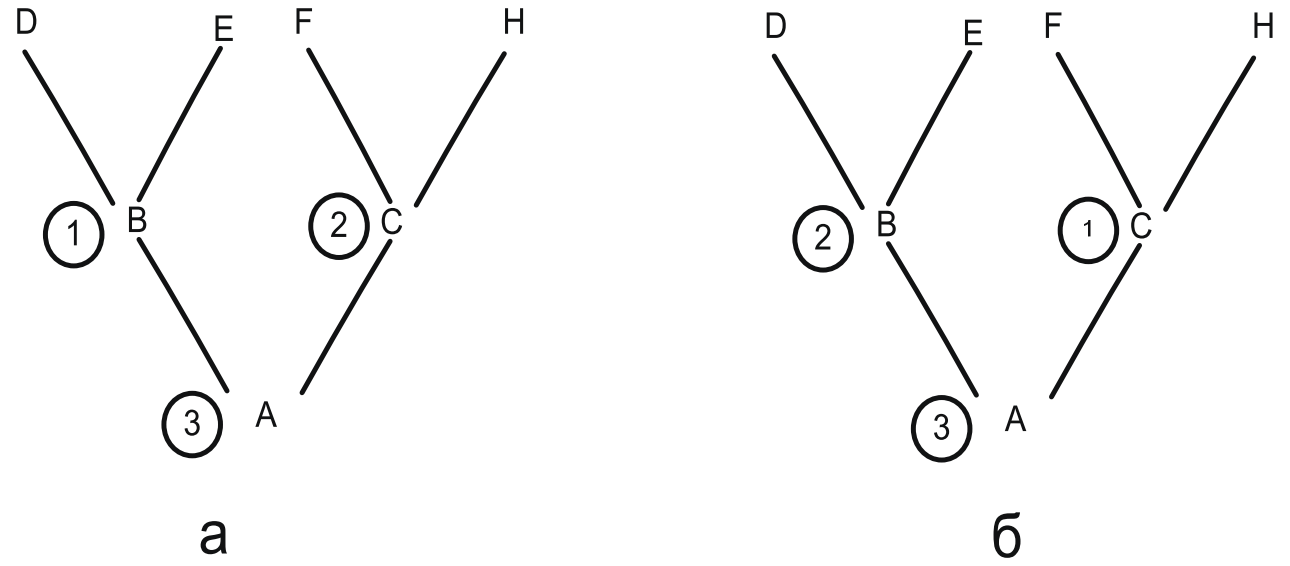


Рис. 2.3. Дерево граматичного розбору слова

Цифри у кружках показують порядок розбору. Результативні дерева ідентичні. Різниця полягає лише в процесі розбору: для *a* - зліва по рядку, для *b* - справа по рядку.

Ця різниця у методі розбору може бути виключена введенням поняття *канонічно впорядкованого граматичного розбору.*

Канонічний вигляд граматичного розбору - це розбір, що застосовується зліва направа по рядку. При цьому в першу чергу розбирається крайня ліва частина пропозиції, якщо це можливо, перш ніж просунутися по рядку вправо для пошуку доступної для розбору ситуації. На рис. 2.3, *а* показано канонічний граматичний розбір пропозиції. Однак канонічно впорядкований граматичний розбір не завжди може бути використаний. Розглянемо приклади:

Приклад 1. Задано граматику

$$A \to x$$
$$A \to Ax \quad \text{(ліва рекурсія)}$$

і рядок xxxx. Канонічний розбір показано на рис. 2.4.

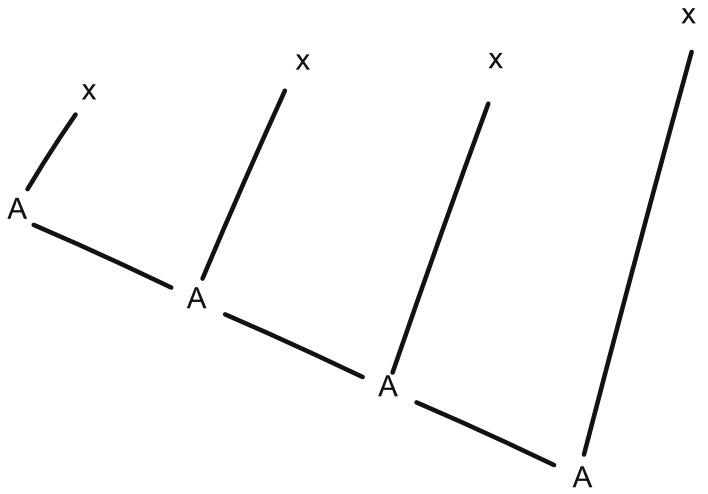


Рис. 2.4. Канонічний розбір лівої рекурсії

Приклад 2. Задано граматику

$$A \to x$$
$$A \to xA \quad \text{(права рекурсія)}$$

і слово xxxx. Граматичний розбір для даного слова не може бути канонічним, оскільки він не може бути виконаний (заводить у тупик). Граматичний розбір у цьому випадку може бути виконаний тільки при розборі правої рекурсії по рядку (рис. 2.5).

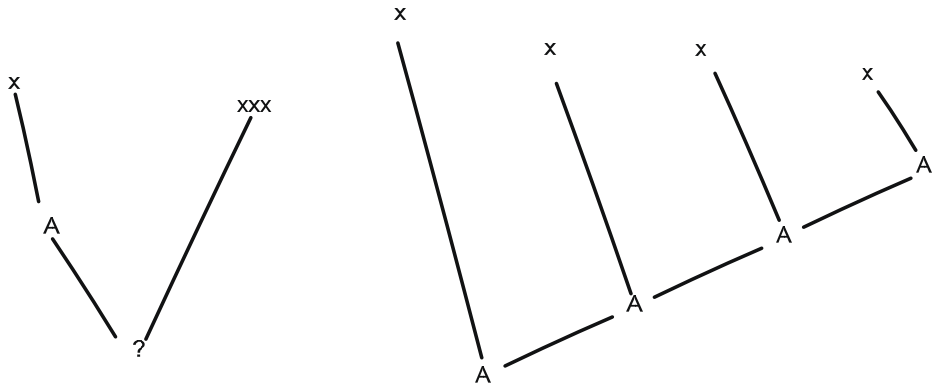


Рис. 2.5. Граматичний розбір правої рекурсії

Приклад 3. Задано граматику

$$A \to x$$
$$A \to xAx \quad \text{(центральна рекурсія)}$$

і слово xxxx. Виконати граматичний розбір. У цьому випадку він не може бути успішно завершений ані при перегляді пропозиції зліва направо (канонічний розбір), ані при перегляді справа наліво. У цьому випадку розбір на кожному етапі починається із середини пропозиції, що розбирається (рис. 2.6).

З розглянутих прикладів 1, 2, 3 можна зробити висновок, що шлях розбору пропозиції визначається типом рекурсії у правилах висновку

граматики. Права рекурсія визначає граматичний розбір, що починається справа.

Ліва рекурсія визначає успішний канонічний вигляд граматичного розбору.

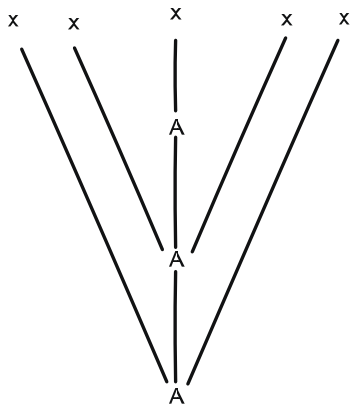


Рис. 2.6. Граматичний розбір центральної рекурсії

Центральна рекурсія визначає граматичний розбір, що починається із середини пропозиції. Але не завжди шлях граматичного розбору може бути легко встановлений за правилами граматики, як це було в наведених прикладах. Розглянемо такий приклад.

Приклад 4. Задано правила граматики у вигляді

$$A \to \omega x;$$

$$B \to Ay;$$

$$C \to Bz \mid \omega D;$$

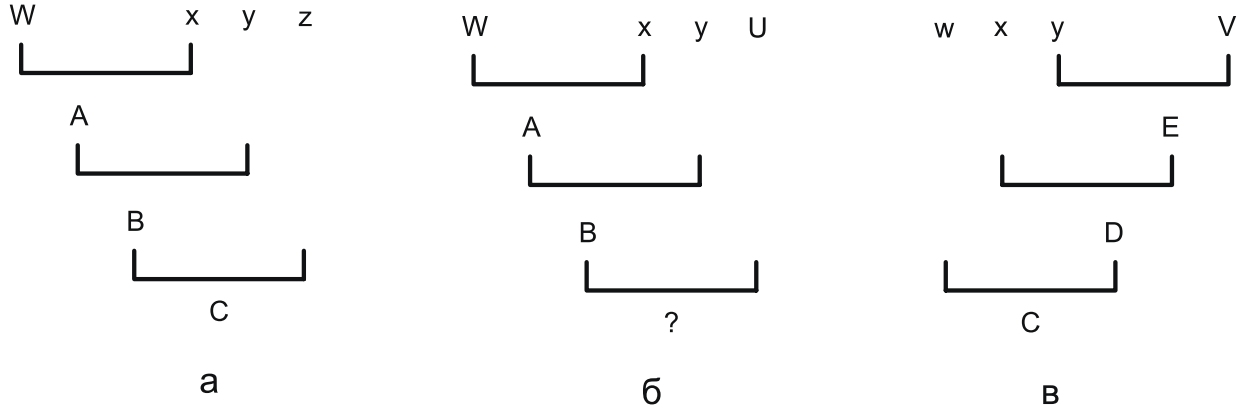
$$D \to xE;$$

$$E \to yv.$$

Необхідно виконати граматичний розбір рядків ωxyz і ωxyv.

Розбір першого рядка може бути успішно виконаний при застосуванні канонічного вигляду розбору.

Результат такого розбору показано на рис. 2.7, а. Використання ж канонічного розбору для другого рядка заводить у тупик (рис. 2.7, б).



а б в

Рис. 2.7. Граматичний розбір рядків ωxyz і ωxyv

Успішне виконання граматичного розбору другого рядка здійснюється за умови, якщо розбір початий з наступної, починаючи зліва, підстроки, тобто з yv. Результати такого розбору показано на рис. 2.7, в. Таким чином, задача граматичного розбору - успішне проведення аналізу пропозицій. Порядок же граматичного розбору залежить від правил висновку (синтаксису) і вигляду аналізованих рядків або пропозицій.

Використовуючи формалізацію як критерій класифікації, усі існуючі методи граматичного розбору можна розділити на ***евристичні*** й ***формалізовані.***

Формалізація методів полягає у систематизації правил, що визначають правильність або помилковість вихідного рядка щодо заданої граматики при застосуванні даного методу на кожному кроці (етапі) граматичного розбору.

Евристичні методи не систематизовані щодо кожного кроку, тобто вони показують, чи правильний даний рядок, тільки лише після остаточного проходження аналізованого тексту.

Евристичний метод відомий ще за назвою методу проб і помилок, пошуку й підстановок, оскільки правильний шлях породжень знаходиться після перевірки всіх можливих шляхів рішення (розбору). Обмеженість використання евристичних методів полягає у такому:

1) правильність або помилковість рядка визначається не відразу на кожному кроці розбору, а лише після закінчення аналізу тексту;

2) правильний шлях породження знаходиться після перевірки всіх можливих шляхів розбору;

3) вибір помилкового шляху потребує зворотного повернення до правильно визначеного останнього стану;

4) при реалізації на машині дуже важко здійснювати зв'язування семантичних правил із синтаксичними.

Усе це негативно позначається на методі з погляду втрати часу.

Разом з тим евристичним методам властиві й деякі переваги: можливість застосування до всіх мов, що робить їх універсальними; орієнтація або «на мету», або «від мети», що визначає два напрямки евристичного методу - «зверху донизу» і «знизу нагору».

Граматичний розбір методом «знизу догори» рядка s мови L, породжуваного граматикою $G =< V_T, V_N, S, P >$, починається з рядка s і полягає у перегляді послідовностей, які одержують у результаті розбору, що веде до S (до мети). Формализовано ціль такого розбору можна подати так: $s => S$, тобто при граматичному розборі визначається, чи є даний рядок пропозицією.

Усі розглянуті раніше приклади граматичного розбору неявно демонстрували цей метод.

Граматичний розбір методом «зверху донизу», який називається ще методом «спуску» (або «рекурсивним спуском»), починається від мети S

(тобто від відправного правила породження або висновку) і далі розглядається послідовність таких породжень, які б призвели до s. Формалізоване подання такого розбору - $S \Rightarrow s$. У результаті граматичного розбору методом «зверху донизу» визначається склад пропозицій мови.

Два ці методи можна подати тими самими деревами породжень, тільки в одному випадку корінь дерева знаходиться внизу, в іншому - угорі. Так, розбір слова ωxyz двома методами для прикладу 4, показано на рис. 2.8.

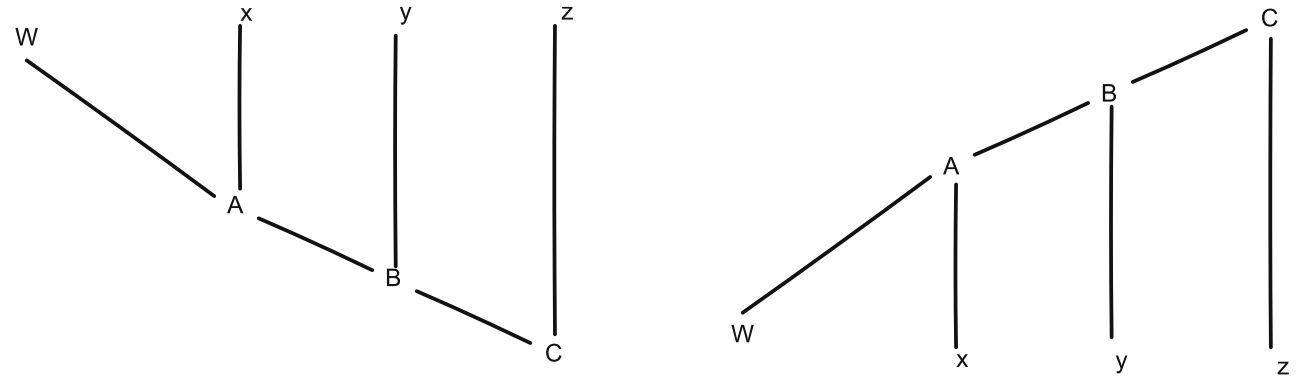


Рис. 2.8. Розбір слова ωxyz

Формалізовані методи з'явилися і з'являються у зв'язку з розробленням трансляторів. При цьому в новітніх методах іноді знаходять висвітлення гарні риси одних і усуваються недоліки інших раніше описаних методів.

Розроблення кожного методу прив'язане до певних класів машин (малого, середнього або більшого типу) і до певних класів мов (залежно від того, з якими мовами працює машина). Оскільки всі мови програмування належать до другого класу за класифікацією Хомського, то й методи в основному орієнтовані на другий клас мов.

Деякі методи контролю вносять обмеження у граматику мови. Звичайно обмеження накладаються на праву частину правил: або за кількістю символів, або за їхнім складом

ПРИКЛАДИ Й ПРАКТИЧНІ ЗАВДАННЯ

2.1. Формальна граматика

Приклад 1. Арифметичні вирази

Розглянемо просту мову, яка визначає обмежену підмножину арифметичних формул, що складаються з натуральних чисел, дужок і знаків арифметичних дій. Варто зазначити, що тут у кожному правилі з лівого боку від стрілки враховується тільки один нетермінальний символ. Такі граматики називаються контекстно-вільними.

Термінальний алфавіт:

$\sum$ = {'0','1','2','3','4','5','6','7','8','9','+','-','*','/','(',')'}.

Нетермінальний алфавіт:

{ ФОРМУЛА, ЗНАК, ЧИСЛО, ЦИФРА }

Правила:

1. *ФОРМУЛА* ⇒ *ФОРМУЛА ЗНАК ФОРМУЛА* (*формула є дві формули з'єднані знаком*)
2. *ФОРМУЛА* ⇒ *ЧИСЛО* (*формула є число*)
3. *ФОРМУЛА* ⇒ (*ФОРМУЛА*) (*формула є формула в дужках*)
4. *ЗНАК* ⇒ + | - | * | / (*знак є плюс або мінус, або помножити або розділити*)
5. *ЧИСЛО* ⇒ *ЦИФРА* (*число є цифра*)
6. *ЧИСЛО* ⇒ *ЧИСЛО ЦИФРА* (*число є число і цифра*)
7. *ЦИФРА* ⇒ 0|*1*|*2*|*3*|*4*|*5*|*6*|*7*|*8*|*9* (*цифра є* 0 *або* 1,*або* ... 9)

Початковий нетермінал:

ФОРМУЛА

Висновок:

Виведемо формулу (12+5) за допомогою зазначених вище правил висновку. Для наочності сторони кожної заміни показані попарно, у кожній парі замінна частина підкреслена:

ФОРМУЛА ⇒ 3(ФОРМУЛА)

(ФОРМУЛА) ⇒ 1(ФОРМУЛА ЗНАК ФОРМУЛА)

(ФОРМУЛА ЗНАК ФОРМУЛА) ⇒ 4(ФОРМУЛА +ФОРМУЛА)

(ФОРМУЛА + ФОРМУЛА) ⇒ 2(Ф(ФОРМУ + ЧИСЛО)

(ФФОРМУЛ + ЧИСЛО) ⇒ 5(Ф(ФОРМУ + ЦИФРА)

(ФФОРМУЛ + ЦИФРА) ⇒ 7(Ф(ФОРМУ + 5)

(ФФОРМУЛ + 5) ⇒ 2(ЧИСЛО + 5)

(ЧИСЛО + 5) ⇒ 6(ЧИСЛО ЦИФРА + 5)

(ЧИСЛО ЦИФРА + 5) ⇒ 5(ЦИФРА ЦИФРА + 5)

(ЦИФРА ЦИФРА + 5) ⇒ 7(1 ЦИФРА + 5)

(1 ЦИФРА + 5) ⇒ 7(1 2 + 5)

Приклад 2. Регулярна граматика

$G = \{N, T, P, S\}$;

$N = \{S\}$;

$T = \{a, b\}$;

$P : S \to a;\ S \to b;\ S \to aS;\ S \to bS.$

За допомогою цієї граматики породжуються рядки символів a і b. Послідовність породження рядків можна пояснити такою схемою:

Застосовуване правило висновку	Зміст рядка
	S
$S \to aS$	aS
$S \to aS$	aaS
$S \to ab$	aab

Результатом висновку є рядок aab.

Приклад 3. Безконтекстна граматика

$$G = \{N, T, P, S\}.$$

$$N = \{S\};$$

$$T = \{a, b\}$$

$$P : S \to aSb;\ S \to ab.$$

За допомогою цієї граматики породжуються рядки вигляду $a^n b^n$.

Приклад висновку рядка $a^3 b^3$ такий:

Застосовуване правило висновку	Зміст рядка
	S
$S \to aSb$	aSb
$S \to aSb$	aaSbb
$S \to ab$	aaabbb

Приклад 4. Безконтекстна граматика більш складного вигляду

$$G = \{N, T, P, S\}.$$

$$N = \{S\};$$

$$T = \{IF, THEN, ELSE, U, B\};$$

$$P : S \to B;$$

$$S \to IF\ U\ THEN\ S;$$

$$S \to IF\ U\ THEN\ S\ \ ELSE\ S.$$

Ця граматика дозволяє формувати різні умовні оператори мови типу Паскаль. Наприклад, оператор IF U THEN IF U THENB ELSE B може бути сформований у такий спосіб:

Застосовуване правило висновку	Зміст рядка
	S
$S \to$ IF U THEN S	IF U THEN S
$S \to$ IF U THEN S ELSE S	IF U THEN S ELSE S
$S \to B$	IF U THEN IF $\bigcup$ THEN B ELSE S
$S \to B$	IF U THEN IF $\bigcup$ THEN B ELSE B

Можливий другий варіант:

Застосовуване правило висновку	Зміст рядка
	S
$S \to$ IF U THEN S ELSE S	IF U THEN S ELSE S
$S \to$ IF U THEN S	IF U THEN IF$\bigcup$ THEN S ELSE S
$S \to B$	IF U THEN IF $\bigcup$ THEN B ELSE S
$S \to B$	IF U THEN IF $\bigcup$ THEN B ELSE B

Вправи

1. Нехай $G=(V_T,V_H,P,S)$ - породжувальна граматика, $V_T=\{a,d,e\}$, де $V_H=\{B,C,S\}$, $P=\{S\to aB,B\to Cd,C\to e\}$. Виписати термінальні ланцюжки, що породжуються даною граматикою, і визначити довжину їхнього виводу.

2. Нехай $G=(V_T,V_H,P,S)$, де $V_T=\{a,d,e\}$, $V_H=\{B,C,S\}$, $P=\{S\to aB,B\to Cd,B\to dC,C\to e\}$. Визначити термінальні ланцюжки, що породжуються даною граматикою, і довжину їхнього виводу.

3. Для граматики G відомі її загальний словник $V=\{A,B,C,D,E\}$ і схема правил $P=\{E\to DCD,E\to A,D\to BC,D\to C,A\to BB\}$. Визначити склад термінального й нетермінального словників, мету граматики, побудувати мову $L(G)$ і визначити довжину висновків для кожного термінального ланцюжка.

4. Визначити, чи є породжувальними такі граматики:

a) $G=(\{A,B\},\{S,D\},P=\{S\to AB,S\to ASD,SD\to B,B\to AS\},S)$;

b) $G=(\{A,B\},\{S\},P=\{S\to ASBAS,S\to AB,AS\to B\},S)$;

c) $G=(\{B,C\},\{A,S\},P=\{S\to A,A\to B,A\to CA\},S)$;

d) $G=(\{B,C\},\{A,S\},P=\{S\to A,A\to B,A\to CAC\},S)$.

5. Дано граматику $G=(V_T,V_H,P,S)$, де $V_T=\{A,B\}$, $V_H=\{S,D\}$, $P=\{S\to AB,S\to ADSB,D\to BSB,DS\to B,D\to\wedge\}$. Показати, що ланцюжок $\mathrm{ABABBAB}$ належить множині $\mathrm{L(G)}$.

6. Дано граматику $G=(\{a,b,c,d,e\},\{A,B,C,D,E\},P=\{A\to ed,B\to Ab,C\to Bc,C\to dD,D\to aE,E\to bc\},C)$. Визначити, чи належить множині $\mathrm{L(G)}$ ланцюжок $\mathrm{eadbcbc}$.

7. Дано $V=\{C,S,a,b\}$ і $V=\{a,b\}$. Визначити, чи є граматикою четвірка $(\mathrm{V_т},\mathrm{V_н},\mathrm{P},\mathrm{S})$ для таких множин правил граматики:

a) $P=\{C\to b,S\to aCb\}$;

b) $P=\{b\to a,C\to Sb\}$;

c) $P=\{C\to bCaC,CS\to\wedge\}$;

d) $P=\{C\to bC,CS\to aS,S\to a\}$.

8. Нехай для кожного $n\geq 1$ $G=(\{a\},\{S\to S^n\},S)$. Показати, що яке б не було $n,L(G_n=\otimes)$.

9. Задано термінальний словник граматики. Визначити граматики, що породжують такі мови:

a) мова $a^nb^na^n$ для $n\geq 1$;

b) мова a^{n2} для $n\geq 1$;

c) мова a^nb^{n2} для $n\geq 1$.

2.2 Контекстно-вільні граматики

Вправи

1. Нехай $V_T = \{a, b\}$, побудувати граматики, що породжують такі мови:

а) мова $L = \{a^n b^n a^n \mid n \geq 1\}$;

б) мова $L = \{a^{n2} \mid n \geq 1\}$;

в) мова $L = \{a^n b^{n2} \mid n \geq 1\}$.

2. Граматики G_1 і G_2 задано такими правилами:

$$P_1 = \{A \to aAbB, A \to bB, A \to A, A \to b,$$
$$B \to bAbb, B \to B, B \to bAb, B \to ab\};$$
$$P_2 = P_1 - \{A \to A, B \to B\}.$$

Показати, що $L(G_1) = L(G_2)$.

3. Дано граматику $G = (V_T, V_H, P, S)$, де $V_T = \{a, b, c\}$, $V_H = \{S, B, S\}$, $P = \{S \to aSBC, S \to aBC, CB \to BC, aB \to ab, bB \to bb, bC \to bc, cB \to cc\}$. Визначити тип граматики і мову, що породжується граматикою.

4. Довести, що кожна лінійна мова породжується граматикою, у якій кожне правило є або ліволінійним, або праволініним.

5. Задано дві граматики G_1 і G_2 правилами вигляду

$$P_1 = \{S \to abS, S \to cA, A \to baA, A \to a\};$$
$$P_2 = \{S \to AB, A \to aAb, B \to aBb, A \to c, B \to c\}.$$

Визначити тип граматик.

6. Довести, що мова $L = \{a^m b^m a^n b^n\}$ не є лінійною.

7. Побудувати приклад КВ-мови, що не є А-мовою, і породжується КС-граматикою, кожне правило якої є або ліволінійним, або праволінійним.

8. Нехай $G = (V_T, V_H, P, S)$, де $V_T = \{a, b, c\}$, $P = \{S \to AB, A \to aAb, B \to aBb, A \to c, B \to c\}$. Показати, що КВ-мова є металінінійною, але не лінійною.

9. Граматику G задано правилами $P = \{S \to aSa, S \to bSb, S \to aaSaa, S \to c\}$. Показати, що вона не є істотно неоднозначною.

10. Мова $L = \{a^n b^m c^r \mid n + m \geq r\}$ є КВ-мовою. Виписати КВ-граматику, яка породжує цю мову. Визначити, до якого більш вузького класу мов вона належить.

2.3 Основні властивості мов

Вправи

1. Дано мову $L = \{aab, aaaab, aaaaaab\}$.

Виконати операції множення, ітерації і транспозиції над ним.

2. Задано словник термінальних символів $V = \{c, b\}$ і мову $L = \{c^n b c^n \mid n \geq 1\}$.

Визначити доповнення мови.

3. Задано мови $L_1 = \{abbc, ab, abcc, ac\}$ і $L_2 = \{xy, xyz, xz, xzy, yz, yzx\}$. Виконати операцію множення цих мов.

4. Задано мову $L = \{a^n b^n \mid n \geq 1\}$. Виконати операції транспозиції, множення, ітерації.

5. Нехай $V_T = \{a, b\}$, $L = \{a^i b^j a^j \mid i, j \geq 1\}$, $M = \{a^j b^j a^i \mid i, j \geq 1\}$, $L_1 = \{a^i b^j a^k \mid i, j, k \geq 1\}$. Показати, що мова $L_2 = L \cap M$ не є контекстно-вільною, а мова $L_3 = L_1 - (L \cap M)$ є контекстно-вільною.

6. Задано мову $L = \{a^n c b^n \mid n \geq 0\}$. Виконати всі можливі операції над даною мовою.

КОНТРОЛЬНІ ЗАПИТАННЯ Й ЗАВДАННЯ ДЛЯ САМОКОНТРОЛЮ

1) Які слова називають рівними?
2) Які складові частини мови?
3) У чому суть формального підходу?
4) Чи пов'язані поява й розвиток формального підходу з необхідністю розв’язання якогось типу задач?
5) Які особливості формальних мов?
6) Які основні конструкції формальних мов?
7) Характеристика необмеженої формальної граматики.
8) У чому суть контекстної формальної граматики?
9) Дайте характеристику безконтекстної формальної граматики.
10) Дайте характеристику регулярної формальної граматики.
11) Що таке граматичний розбір?
12) Які основні типи породжувальних граматик?
13) Класи формальних граматик.
14) Основне визначення контекстно-вільних граматик.
15) Чи можна подати у вигляді графа взаємозв'язок класів граматик?
16) Які властивості формальних мов?

ТЕМИ ДЛЯ САМОСТІЙНОЇ РОБОТИ

1) Метамова Хомського - Щутценберже.
2) Форми Бекуса - Наура (БНФ).
3) Приклади опису ідентифікатора з використанням БНФ.
4) Приклади опису ідентифікатора.
5) Діаграми Вірта.
6) Приклади опису ідентифікатора з використанням діаграм Вірта.
7) Визначення й структура розпізнавача.
8) Елементарні конструкції.
9) Приклади елементарних конструкцій.

3. СКІНЧЕННІ АВТОМАТИ І ЇХНІЙ ЗВ'ЯЗОК З МОВАМИ І ГРАМАТИКАМИ

3.1. ЗАГАЛЬНЕ ВИЗНАЧЕННЯ СКІНЧЕННОГО АВТОМАТА

Скінченним автоматом (СА) називається п'ятірка $A=(N,T,P,S,F)$, де N - скінченна множина станів автомата;

T - вхідний алфавіт - скінченна множина символів;

P - функція переходів автомата;

S - початковий стан, $S \in N$;

F - множина скінченних станів, $F \subseteq N$.

Спочатку автомат перебуває у стані S. На вхід СА надходять символи, що належать вхідному алфавіту. Послідовність вхідних символів утворить вхідний ланцюжок. Перебуваючи в деякому стані й одержавши на вхід черговий символ, автомат переходить у наступний стан, зумовлений значенням функції переходів.

У загальному випадку функція переходів для даної пари символів стану може визначати кілька варіантів переходу. У такому випадку автомат називають недетермінованим (НСА).

Якщо, прочитавши вхідний ланцюжок α, автомат зупинився в деякому стані *B*, то говорять, що α переводить автомат у стан *B*. Якщо *B* - один з кінцевих станів, тобто $B \in F$, то СА допускає ланцюжок α.

Множину всіх прийнятих автоматом ланцюжків утворить мова $L(A)$, прийнята автоматом.

Мова, яка породжена автоматною граматикою *G*, збігається з мовою, прийнятою відповідним скінченним автоматом:

$$L(G)=L(A). \qquad (3.1)$$

СА може задаватися за допомогою діаграми переходів. Наприклад, граф автоматної граматики G_8 може вважатися діаграмою переходу A_8.

При переході від автоматної граматики до СА у загальному випадку одержують недетермінований СА, що утрудняє його використання як розпізнавача автоматної мови. Недетермінованість автомата визначається тим, що для деяких вершин його діаграми переходів є кілька дуг, які виходять із цих вершин і позначені тим самим символом.

Для усунення неоднозначності НСА переводять у ДСА (детермінований СА).

Найбільш простою моделлю є автомат зі скінченним числом станів (з кінцевою пам'яттю) - скінченний автомат.

Детермінованим скінченним автоматом називається упорядкована система з п'яти об'єктів - «упорядкована п'ятірка»:

$$A=(X,S,s_0,\delta,F),$$

де $X = \{x_1, x_2, ..., x_r\}$ - множина вхідних символів (вхідний алфавіт); $S = \{s_1, s_2, ..., s_r\}$ - множина внутрішніх станів; r, n - скінченні; $s_0 \in S$ - початковий стан; δ - функція, що відображує $S \times X$ у S, і звичайно записується так: $\delta : S \times X \to S$. Ця функція однозначно ставить у відповідність парі символів (s_i, x_j) деякий символ $s_k \in S$, при цьому $F \subseteq S$ - деяка виділена підмножина станів автомата (заключні стани).

Цей автомат можна інтерпретувати так, як це показано на рис. 3.1.

Автомат складається з керуючого пристрою, що зчитує інформацію, і нескінченної вправо вхідної стрічки, поділеної на клітинки. Спочатку на стрічці записаний вхідний ланцюжок так, що в кожній клітинці стрічки утримується по одному символу ланцюжка. Початковий символ записаний у крайній лівій клітинці, а всі клітинки тієї частини стрічки, що розташована праворуч останнього символу запису вхідного ланцюжка, порожні, тобто в кожній із цих клітинок записаний «порожній» символ A. Пристрій у початковий момент розташовано проти крайньої лівої клітинки стрічки, а керуючий пристрій перебуває у початковому стані:

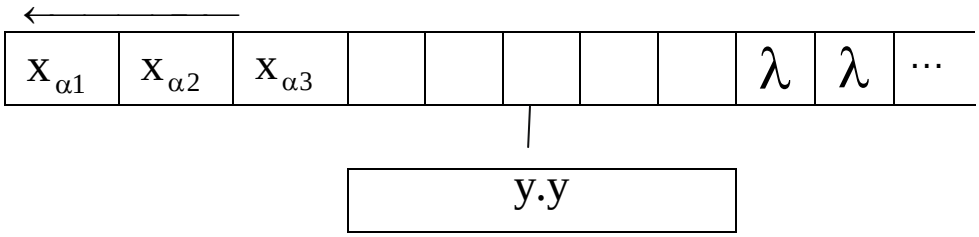


Рис. 3.1. Детермінований скінченний автомат

У кожний черговий такт головка сприймає символ на стрічці, керуючий пристрій змінює свій стан відповідно до відображення δ, а стрічка зрушується на одну клітинку вліво. Якщо пристрій сприймає порожній символ $\wedge$, що записаний праворуч від останнього символу вхідної послідовності, то це буде означати припинення роботи автомата, тобто стан керуючого пристрою більше не змінюється.

Припустимою, або прийнятною, називається вхідна послідовність, що має таку властивість: коли головка сприймає останній символ цієї вхідної послідовності, автомат переходить в один зі станів множини F.

Множина припустимих вхідних послідовностей автомата A або множина послідовностей, поданих в автоматі A множиною станів F, позначається $L(A)$. Така множина, будучи множиною ланцюжків символів зі скінченного алфавіту X, з погляду теорії мов буде деякою мовою над термінальним словником $V_T = X$. Таким чином, природно виникає спосіб зіставлення автоматів, мов і граматик.

Для скінченних автоматів множина $L(A)$ визначається відомою теоремою Кліні.

Теорема 4.1 *(Кліні).* Для будь-якого скінченного автомата A множина L(A) припустимих послідовностей є регулярною, тобто мовою типу 3. Ця теорема пояснює, чому мови типу 3 називаються мовами зі скінченним числом станів.

Детермінованим скінченним автоматом з двома стрічками називається впорядкована шістка

$$A = (X, Y, S, s_0, \delta, \lambda), \tag{3.2}$$

де X, S, s_0 і δ мають той же зміст, що й для скінченного автомата з однією стрічкою; $Y = \{y_1, y_2, ..., y_l\}$ - множина вихідних символів (вихідний алфавіт); l - скінченне; λ - функція, що здійснює відображення $S \times X$ у Y, тобто $\lambda : S \times X \to Y$.

Інтерпретацію цього автомата показано на рис. 3.2.

Крім вхідної стрічки автомат має нескінченну вправо вихідну стрічку, що може переміщуватися тільки в один бік - праворуч або ліворуч. Кожний черговий такт у клітинці стрічки друкується символом, і стрічка зрушується на одну клітинку. Цей автомат називається послідовною машиною, або автоматом Мілі. Автомат Мілі M кожному ланцюжку вхідних символів u однозначно ставить у відповідність ланцюжок вихідних символів ω, що записується як $M(u) = \omega$. Очевидно, що $M(\wedge) = \wedge$.

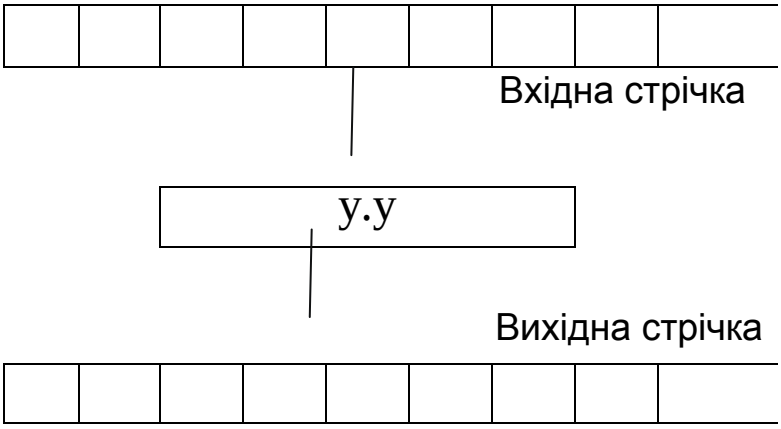


Рис. 3.2. Детермінований скінченний автомат з двома стрічками

Недетермінованим скінченним автоматом з однією стрічкою називається упорядкована п'ятірка $A = (X, S, S_0, \delta, F)$, де X, S, F мають той же зміст, що й у визначенні детермінованого автомата; δ - відображення множини $S \times X$ у множину всіх підмножин станів S; S_0 - деяка виділена (початкова) підмножина множини S.

Подання множини вхідних послідовностей у недетермінованому автоматі розуміється так. Множина вхідних послідовностей U припустима

в недетермінованому скінченному автоматі, якщо для кожної послідовності цієї множини знайдуться два таких стани $s_i \in S_0$ і $s_i \in F$ і такий варіант роботи (тобто такі конкретні значення функції δ), що послідовність U переводить автомат зі стану s_i у стан s_j.

Доведено таку теорему.

Теорема 4.2 *(Рабіна і Скотта).* У недетермінованих однострічкових скінченних автоматах так само, як і в детермінованих, припустимі регулярні множини ланцюжків і тільки вони. За допомогою недетермінованих автоматів не можна розширити клас поданих множин. Однак недетермінований скінченний автомат звичайно має менше число станів порівняно з детермінованим автоматом, що представляє ту ж множину.

Скінченний автомат із двома стрічками є недетермінованим, якщо при одній і тій же ситуації, що характеризується парою (s_i, x_j), можливо скінченне число варіантів його дії. Отже, такий автомат може одному вхідному ланцюжку поставити у відповідність скінченну множину вихідних ланцюжків.

Двосторонній скінченний автомат відрізняється від звичайного детермінованого автомата тим, що вхідна стрічка нескінченна в обидва боки й може рухатися не тільки вліво, але й вправо, а також може залишатися нерухомою. При цьому відображуюча функція $\delta : S \times X \to S$ замінюється функцією $\delta : S \times X \to S \times d$, де $d = \{-1,0,+1\}$. Символ d показує, у який бік повинна зрушитися вхідна стрічка: -1 відповідає зрушенню на одну клітинку вправо, $+1$ - зрушенню вліво на одну клітинку, 0 - стрічка залишається нерухомою.

Множина $L(A)$ усіх припустимих послідовностей такого автомата визначається такою теоремою.

Теорема 4.3 (*Рабіна, Скотта, Шепердсона).* Для кожного двостороннього автомата A можна ефективно визначити такий автомат B, що має ту ж множину прийнятних вхідних послідовностей, що й автомат A. Тому множина $L(A)$ регулярна, тобто є мовою типу 3.

Таким чином, двосторонній автомат не розширює можливості скінченних автоматів.

3.2. АВТОМАТ МІЛІ І АВТОМАТ МУРА

Автомат Мілі (англ. Mealymachine) - скінченний автомат, вихідна послідовність якого (на відміну від автомата Мура) залежить від стану автомата та вхідних сигналів. Це означає, що в графі станів кожному ребру відповідає деяке значення (вихідний символ). У вершині графа автомата Мілі записуються вихідні сигнали, а дугам графа приписують

умову переходу з одного стану в інший, а також вхідні сигнали. Автомат Мілі можна описати п'ятіркою

$$(Q, X, Y, f, g), \qquad (3.3)$$

де Q - множина станів автомата; X - множина вхідних символів; Y - множина вихідних символів; $q = f(Q, X)$ - функція станів; $y = g(Q, Y)$ - функція вихідних символів.

Закон функціонування автомата Мілі задається такими рівняннями:

$$a(t+1) = \delta(a(t),\ z(t));\ \ w(t) = \lambda(a(t), z(t),\ t = 0,\ 1,\ 2,...),$$

а закон функціонування автомата Мура - такими:

$$a(t+1) = \delta(a(t),\ z(t));\ \ w(t) = \lambda(a(t)),\ t = 0,\ 1,\ 2,...).$$

З порівняння законів функціонування видно, що, на відміну від автомата Мілі, вихідний сигнал в автоматі Мура залежить тільки від поточного стану автомата і у явному вигляді не залежить від вхідного сигналу. Для повного задання автомата Мілі або Мура додатково до законів функціонування необхідно вказати початковий стан і визначити внутрішній, вхідний і вихідний алфавіти. Автомати Мілі - автомати 1-го роду, R-авт., автомати Мура - автомати 2-го роду, S-авт., $C = R + S$ - комбіновані автомати.

3.2.1. Синтез автомата Мілі

На етапі одержання граф-схеми алгоритму входи вершин, що випливають за операторними, позначають символами $a_1, a_2, ...$ за такими правилами:

1) символом a_1 позначають вхід вершини, що випливає з початкової, а також вхід кінцевої вершини;

2) входи всіх наступних за операторними вершин, які повинні бути позначені;

3) входи різних вершин, за винятком кінцевої, позначають різними символами;

4) якщо вхід вершини позначають, то тільки одним символом.

Для проведення індексування буде потрібно скінченне число символів $a_1,...,a_m$. Результатом першого етапу є зазначена граф-схеми алгоритму, що є основою для другого етапу - переходу до графа або таблиць переходів-виходів.

На другому етапі на основі отриманої граф-схеми алгоритму будують граф автомата або таблиці переходів-виходів. Для цього вважають, що в автоматі буде стільки станів, скільки символів a_i знадобилося при оцінюванні граф-схеми алгоритму.

На площині рисунка зазначено всі стани автомата a_i. Для кожного зі станів a_i відповідно до граф-схеми алгоритму показано всі шляхи, що ведуть в інші стани й проходять обов'язково тільки через одну операторную вершину.

На підставі зазначеної граф-схеми алгоритму можна побудувати таблицю переходів-виходів. Для мікропрограмних автоматів таблиця переходів-виходів будується у вигляді списку, при цьому пряма й зворотна таблиці розрізняються. Для даного автомата пряму таблицю надано в табл. 3.1, зворотну - у табл. 3.2.

Таблиця 3.1

Пряма таблиця переходів-виходів граф-схеми алгоритму

Am	As	X	Y
a1	a2	1	y_1, y_3
a2	a5	$\overline{x_2}$	y_6
	a7	x_2	y_4
a3	a4	1	y_2
a4	a5	$\overline{x_5}$	y_6
	a6	x_5	y_7, y_{10}
a5	a6	1	y_7, y_{10}
a6	a8	$\overline{x_4}$	y_2
	a9	x_4	y_2, y_4
a7	a9	1	y_2, y_4
a8	a10	1	y_3, y_6
a9	a11	1	y_7
a10	a3	$\overline{x_5}, x_6$	y_3, y_6
	a11	x_5	y_7
	a12	$\overline{x_5}, \overline{x_6}$	y_8
a11	a12	x_1	y_8
	a13	$\overline{x_1}$	y_1, y_9
a12	a4	$\overline{x_2}$	y_2
	a16	x_2	y_2, y_4
a13	a16	1	y_2, y_4
a14	a15	1	y_3, y_6
a15	a17	x_5	y_7
	a18	$\overline{x_5}, \overline{x_6}$	y_8
	a20	$\overline{x_5}, x_6$	y_3, y_6

Закінчення табл. 3.1

Am	As	X	Y
a16	a17	1	y_7
a17	a18	x_1	y_8
	a19	$\overline{x_1}$	y_1, y_9
a18	a21	$\overline{x_2}$	y_3, y_4
	a22	x_2	y_6, y_9
a19	a21	1	y_3, y_4
a20	a22	1	y_6, y_9
a21	a23	1	y_8
a22	a1	$x_6, \overline{x_5}$	y_1, y_9
	a14	$\overline{x_6}, x_1$	y_2
	a24	$\overline{x_6}, \overline{x_1}$	y_7
a23	a1	$\overline{x_5}$	y_1, y_9
	a24	x_5	y_7
a24	a15	$\overline{x_2}$	y_3, y_6
	a1	x_2	-

Таблиця 3.2

Зворотна таблиця переходів-виходів граф-схеми алгоритму

Am	As	X	Y
a22	a1	$x_6, \overline{x_5}$	y_1, y_9
a23		$\overline{x_5}$	y_1, y_9
a24		x_2	-
a1	a2	1	y_1, y_3
a10	a3	$\overline{x_5}, x_6$	в3,в6
a3	a4	1	y_2
a12		$\overline{x_2}$	y_2
a2	a5	$\overline{x_2}$	y_6
a4		$\overline{x_5}$	y_6
a4	a6	x_5	y_7, y_{10}
a5		1	y_7, y_{10}
a2	a7	x_2	y_4
a6	a8	$\overline{x_4}$	y_2

Закінчення табл. 3.2

Am	As	X	Y
a6	a9	x_4	y_2, y_4
a7		1	y_2, y_4
a8	a10	1	y_3, y_6
a9	a11	1	y_7
a10		x_5	y_7
a10	a12	$\overline{x_5}, \overline{x_6}$	y_8
a11		x_1	y_8
a11	a13	$\overline{x_1}$	y_1, y_9
a22	a14	$\overline{x_6}, x_1$	y_2
a14	a15	1	y_3, y_6
a24		$\overline{x_2}$	y_3, y_6
a12	a16	x_2	y_2, y_4
a13		1	y_2, y_4
a15	a17	x_5	y_7
a16		1	y_7
a15	a18	$\overline{x_5}, \overline{x_6}$	y_8
a17		x_1	y_8
a17	a19	$\overline{x_1}$	y_1, y_9
a15	a20	$\overline{x_5}, x_6$	y_3, y_6
a18	a21	$\overline{x_2}$	y_3, y_4
a19		1	y_3, y_4
a18	a22	x_2	y_6, y_9
a20		1	y_6, y_9
a21	a23	1	y_8
a22	a24	$\overline{x_6}, \overline{x_1}$	y_7
a23		x_5	y_7

3.2.2. Синтез автомата Мура

Для автомата Мура на етапі одержання граф-схеми алгоритму розмітку проводять за такими правилами:

1) символом a_1 позначають початкову й кінцеву вершини;

2) різні операторні вершини позначають різними символами;

3) усі операторні вершини повинні бути позначені.

Таблицю переходів-виходів автомату Мура наведено в табл. 3.3 (пряма). Звичайно для автомата Мура в таблиці переходів-виходів додатковий стовпець для вихідних сигналів не використовується і вихідний сигнал записується в стовпці, де вказується вихідний стан am або стан переходу *a*.

Таблиця 3.3

Переходи-виходи автомата Мура

Am(y)	As	X
a1(--)	a2	1
a2(y_1,y_3)	a4	x_2
	a5	$\overline{x_2}$
a3(y_2)	a5	$\overline{x_5}$
	a6	x_5
a4(y_4)	a7	1
a5(y_6)	a6	1
a6(y_7,y_{10})	a7	x_4
	a8	$\overline{x_4}$
a7(y_2,y_4)	a10	1
a8(y_2)	a9	1
a9(y_3,y_6)	a10	x_5
	a12	$\overline{x_5}$,$\overline{x_6}$
	a13	$\overline{x_5}$,x_6
a10(y_7)	a11	$\overline{x_1}$
	a12	x_1
a11(y_1,y_9)	a14	1
a12(y_8)	a14	x_2
	a3	$\overline{x_2}$
a13(y_3)	a3	1
a14(y_2,y_4)	a16	1

Закінчення табл. 3.3

Am(y)	As	X
a15(y_3,y_6)	a16	x_5
	a18	$\overline{x_5}$,$\overline{x_6}$
	a19	$\overline{x_5}$,x_6
a16(y_7)	a17	$\overline{x_1}$
	a18	x_1
a17(y_1,y_9)	a20	1
a18(y_8)	a20	x_2
	a22	$\overline{x_2}$
a19(y_3)	a22	1
a20(y_3,y_4)	a21	1
a21(y_8)	a23	$\overline{x_5}$
	a24	x_5
a22(y_6,y_9)	a23	x_6,$\overline{x_5}$
	a24	$\overline{x_6}$,$\overline{x_1}$
	a25	$\overline{x_6}$,x_1
a23(y_1,y_9)	a1	1
a24(y_7)	a1	x_2
	a15	$\overline{x_2}$
a25(y_2)	a15	1

3.2.3. Перетворення автомата Мілі в автомат Мура

Між автоматами Мілі і Мура існує відповідність, що дозволяє перетворити закон функціонування одного з них в інший або навпаки (рис. 3.3). Автомат Мура можна розглядати як окремий випадок автомата Мілі, маючи на увазі, що послідовність станів виходів автомата Мілі випереджає на один такт послідовність станів виходів автомата Мура, тобто розходження між автоматами Мілі і Мура полягає у тому, що в автоматах Мілі стан виходу виникає одночасно з відповідним йому станом входу, а в автоматах Мура - із затримкою на один такт в автоматах Мура вхідні сигнали змінюють тільки стан автомата.

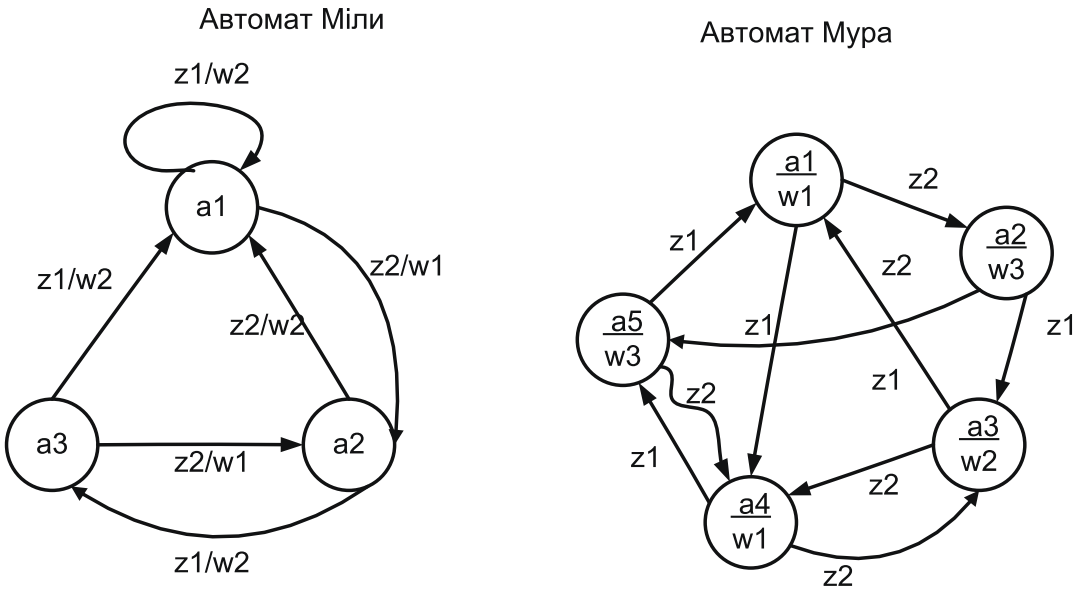


Рис. 3.3. Вигляд автоматів Мілі і Мура

Нехай необхідно перетворити автомат Мілі в автомат Мура. Граф автомата Мілі показано на рис 3.4.

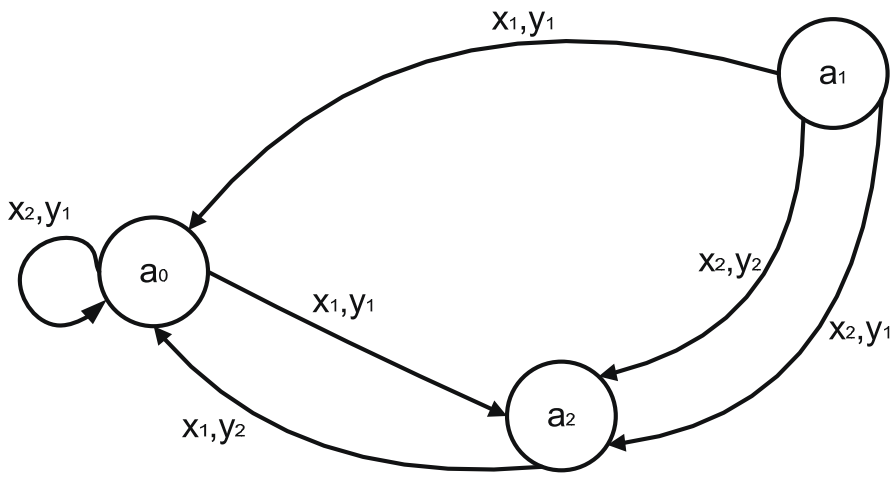


Рис. 3.4. Граф автомата Мілі

В автоматі Мілі $X_a = \{x_1, x_2\}, Y_a = \{y_1, y_2\}, A_a = \{a_0, a_1, a_2\}$.

В еквівалентному автоматі Мура $X_b = X_a = \{x_1, x_2\}$, $Y_b = Y_a = \{y_1, y_2\}$.

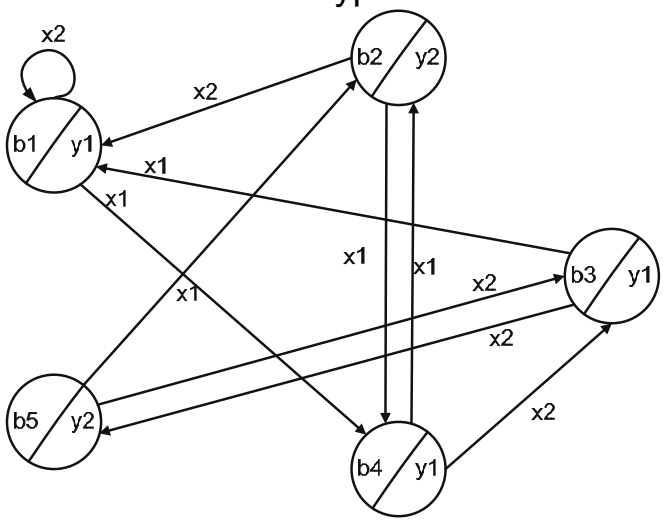
Побудуємо множину станів A_b автомата Мура, для чого знайдемо множини пар, що породжуються кожним станом автомата S_a:

Стан	Породжувані пари
a_0	$\{(a_0, y_1), (a_0, y_2)\} = \{b_1, b_2\}$
a_1	$\{(a_1, y_1)\} = \{b_3\}$
a_2	$\{(a_2, y_1), (a_2, y_2)\} = \{b_4, b_5\}$

Звідси маємо множини A_b станів автомата Мура $A_b = \{b_1, b_2, b_3, b_4, b_5\}$. Для знаходження функції виходів L_b з кожним станом, що являє собою пари вигляду (a_i, y_g), ототожнимо вихідний сигнал, що є другим елементом цієї пари. У результаті цього маємо

$$b(b_1) = l_b(b_3) = l_b(b_4) = y_1;\ b(b_2) = l_b(b_5) = y_2.$$

Побудуємо функцію переходів d_b. Оскільки в автоматі S_a зі стану a_0 є перехід під дією сигналу x_1 у стан a_2 з видачею y_1, то із множини станів $\{b_1, b_2\}$, що породжуються a_0, в автоматі S_b повинен бути перехід у стан $(a_2, y_1) = b_4$ під дією сигналу x_1. Аналогічно з $\{b_1, b_2\}$ під дією x_2 повинен бути перехід у $(a_0, y_1) = b_1$, з $(a_1, y_1) = b_3$ під дією x_1 - у $(a_0, y_1) = b_1$, а під дією x_2 - у $(a_2, y_2) = b_5$ і, нарешті, зі станів $\{(a_2, y_1), (a_2, y_2)\} = \{b_4, b_5\}$ під дією x_1 - перехід у $(a_0, y_2) = b_2$, а під дією x_2 - у $(a_1, y_1) = b_3$. У результаті цього маємо граф (рис. 3.5) і таблицю переходів еквівалентного автомата Мура.



y_g	y_1	y_2	y_1	y_1	y_2
$x_j \backslash b_j$	b_1	b_2	b_3	b_4	b_5
x_1	b_4	b_4	b_1	b_2	b_2
x_2	b_1	b_1	b_5	b_3	b_3

Рис. 3.5. Граф еквівалентного автомата Мура

Як початковий стан автомата S_b можна взяти кожне зі станів b_1 або b_2, оскільки обидва вони породжені станом a_0 автомата S_a.

3.2.4. Зв'язок між моделями автоматів Мілі і Мура

Два автомати з однаковими вхідними й вихідними алфавітами називаються еквівалентними, якщо після установлення їх у початковий стан їхні реакції на будь-яке вхідне слово збігаються.

Незважаючи на те що автомати функціонують по-різному, завжди можна побудувати автомат однієї моделі, що є еквівалентним автомату іншої моделі в тому розумінні, що їхні реакції на одні й ті ж вхідні ланцюжки будуть однакові. Загальний підхід до побудови еквівалентних автоматів:

1) нехай задано автомат Мура, який необхідно перетворити в еквівалент автомат Мілі:

$S_a = (A_a, Z_a, W_a, \delta_a . \lambda_a, a_1 A)$ - автомат Мура, $S_b = (A_b, Z_b, W_b, \delta_b . \lambda_b, a_1 B)$ - автомат Мілі.

2) зажадаємо, щоб $A_b = A_a$, $Z_b = Z_a$, $W_b = W_a$, $\delta_b = \delta_a . \lambda_a$, $a_1 B = a_1 A, \lambda_b - ?$.

Таким чином, для переходу від автомата Мура до автомата Мілі необхідно й достатньо у графі автоматата Мура винести символ вихідного алфавіту з розглянутої вершини й приписати його всім дугам, що входять у цю вершину. В еквівалентному автоматі Мілі кількість станів така ж, як і у вихідному автоматі Мура.

Перехід від автомата Мілі до еквівалентного йому автомата Мура більш складний. Це пов'язане з тим, що в автоматі Мура в кожному стані виробляється тільки один вихідний сигнал. Єдиним обмеженням, що накладається на можливість такого переходу, є те, що вихідний автомат Мілі не повинен мати недосяжних станів:

1) нехай задано автомат Мілі, який необхідно перетворити в еквівалент - автомата Мура:

$S_a = (A_a, Z_a, W_a, \delta_a . \lambda_a, a_1 A)$ - автомат Мілі, $S_b = (A_b, Z_b, W_b, \delta_b . \lambda_b, a_1 B)$ - автомат Мура. Зажадаємо, щоб $Z_b = Z_a, W_b = W_a$;

2) визначимо множину станів Мура A_b, для цього кожному стану $A_S \in A_a$ ставиться у відповідність множина A_S, що являє собою всілякі пари вигляду (a_S, w_g), де w_g - вихідні сигнали, проставлені уздовж дуг автомата Мілі, що входять у вершину a_S.

Множина A_b - це є об'єднання $A_S = A_b$, $S \in (1, M)$. У загальному випадку кількість вершин в автоматі Мура більша, ніж в автоматі Мілі. Функцію виходу λ_b і функцію переходу W_b слід визначити таким чином: кожному стану автомата Мура, надається пара (a_S, w_g), яка ставиться у відповідність вихідному сигналу w_g. Якщо в автоматі Мілі S_a був перехід з вершини a_m під дією сигналу z_f у вершину a_S, тобто $S_a(a_m, z_f) = a_S$ і при цьому автомат виробляє сигнал $\lambda_a(a_m, z_f) = w_k$, то в автоматі Мура буде перехід із множини станів $S_a(a_m, z_f) = a_S$, у w_k під дією того ж вхідного сигналу z_f.

3.3. МОЖЛИВОСТІ ПОДАННЯ ФОРМАЛЬНИХ ГРАМАТИК У ВИГЛЯДІ СКІНЧЕННИХ АВТОМАТІВ

Відповідність між основними видами автоматів і відповідними мовами подано в табл. 3.4. У цій таблиці використано такі позначення. Стрілка $\longleftrightarrow$ означає, що мова тоді й тільки тоді є типом $i\,(i=0,\ 1,\ 2,\ 2^D,\ 3)$, коли вона може бути подана у вигляді, автомата типу $A_k\,(k=1,\ 2,...,6)$. Стрілка $-\!\!\rightarrow$ означає, що якщо мова подана в автоматі A_k, то вона є мовою типу i. Стрілка $\leftarrow\!\!-$ означає, що якщо мова є мовою типу i, то знайдеться автомат типу A_k, у якому вона подана.

Таблиця 3.4

Відповідність між основними видами автоматів і відповідними мовами

№ п/п	Найменування автомата	Тип поданої мови
1	Детермінований і недетермінований скінченні автомати	$\longleftrightarrow 3$
2	Детермінований автомат з магазинною пам'яттю	$\longleftrightarrow 2$
3	Недетермінований автомат з магазинною пам'яттю	$\longleftrightarrow 2$
4	Детермінований лінійно обмежений автомат	$\leftarrow\!\!- 2$ $-\!\!\rightarrow 1$
5	Недетермінований лінійно обмежений автомат	$\longleftrightarrow 1$
6	Детермінована й недетермінована машини Тьюринга	$\longleftrightarrow 0$

Розглянемо можливе трактування понять і результатів загальної теорії мов стосовно трьох основних областей, де ці результати використовуються: математичної лінгвістики, мов програмування й теорії автоматів.

У табл. 3.5 зібрано аналогічні поняття із цих трьох областей і дано відповідні їм поняття загальної теорії мов.

Таблиця 3.5

Основні поняття загальної теорії мов

Позначення	Загальна теорія мов	Математична лінгвістика	Програмування	Математичні моделі
$V_т$	Термінальний словник (алфавіт)	Основний словник мови	Вихідні символи	Вихідний алфавіт
$V_н$	Нетермінальний словник (алфавіт)	Допоміжні граматичні терміни	Набір команд	Множина станів керуючого пристрою
S	Початковий нетермінальний символ	Пропозиції	Програма	Початковий стан
P	Система продукций	Синтаксичні правила	Операції	Відображувальна функція

ПРИКЛАДИ Й ПРАКТИЧНІ ЗАВДАННЯ

3.1. Скінченні автомати

Вправи

1. За кожним із скінченних автоматів A, наведених нижче, побудувати праволінійну КВ-граматику, що породжує *$T(A)$-множину:*

а) $A = (S, X, p, \delta\{p_3\}), \delta$ (задається в табл. 3.6);

б) $A = (S, X, p_1, \delta\{p_4, p_5\}), \delta$ (задається в табл. 3.7).

Таблиця 3.6

Скінченний автомат, що породжується *$T(A)$ множиною* вигляду $A = (S, X, p, \delta\{p_3\}), \delta$

Позначення	а	в
P_1	P_2	P_5
P_2	P_3	P_4
P_3	P_2	P_5
P_4	P_3	P_1
P_5	P_2	P_4

Таблиця 3.7

Скінченний автомат, що породжується T(A)-*множиною* вигляду $A = (S, X, p_1, \delta\{p_4, p_5\}), \delta$

Позначення	а	в	с
P_1	P_2	P_3	P_1
P_2	P_1	P_6	P_6
P_3	P_4	P_4	P_2
P_4	P_2	P_3	P_1
P_5	P_4	P_6	P_3
P_6	P_1	P_2	P_6

На перетинанні рядка p_i і стовпця x указано значення $\delta(p_i, x)$.

2. Побудувати МП-автомат M_1 такий, що $X = \{a, b\}$ і $T(M)$- множина всіх ланцюжків, що містять однакове число вхідних символів a і b.

3. Побудувати МП-автомат M, такий, що $X = \{a,b\}$ і $T(M) = \{a^n b^n a^i \mid n \geq 1, i \geq 1\} U \{a^n b^{2n} a^i b^{3i} \mid n \geq 1, i \geq 1\}$.

4. Побудувати МП-автомат, що допускає мову, породжену граматикою із правилами $P = \{S \to TT, T \to Ta, T \to bT, T \to c\}$.

5. Задано множину команд скінченного автомата, що допускає мову $\{a^n b^n \mid n,m \geq 0\}$:

$$(a_1 S_0) \to S_1;$$
$$(a_1 S_1) \to S_1;$$
$$(b_1 S_1) \to S_2;$$
$$(b_1 S_2) \to S_2;$$
$$(b_1 S_2) \to S_0.$$

Побудувати граматику, яку породжує ця мова, і визначити її тип.

6. Побудувати лінійно-обмежені автомати, що допускають такі мови:

$$L_1 = \{a^n b^n a^n \mid n \geq 0\};$$
$$L_2 = \{xcx \mid x \in \{a,b\}*\}.$$

Мова $L = \{a^p b^q c^r \mid p + q \geq r\}$ є КВ-мовою.

Виписати КВ-граматику, яку породжує ця мова. Визначити, до якого більш вузького класу мов вона належить. Побудувати детермінований МП-автомат, який допускає ця мова.

7. Які з наведених множин послідовностей можуть бути розпізнані скінченним автоматом:

а) множина всіх послідовностей: 0, 1, 00, 01, 10, 11, 000, 001, 010, ... ;

б) числа 1, 2, 4, 8, ... , 2n, ... , записані у двійковій системі числення;

в) та ж сама множина чисел, записаних в унарному коді: 1, 11, 1111, 11111111, 1111111111111111, ... ;

г) множина послідовностей, у яких число нулів дорівнює числу одиниць;

д) послідовності: 0, 101, 11011, ... , 1k01k (k - число повторень одиниці)?

3.2. Автомат Мілі і автомат Мура

Будь-яку скінченну множину слів $E = \{a_1,...,a_k\}$ подано в автоматі. Ідея побудови автомата за скінченною множиною слів ілюструється графом на рис. 3.6, а, де заключні стани $q_{n-k},...,...,q_{n-1}$ зображено подвійним кружком.

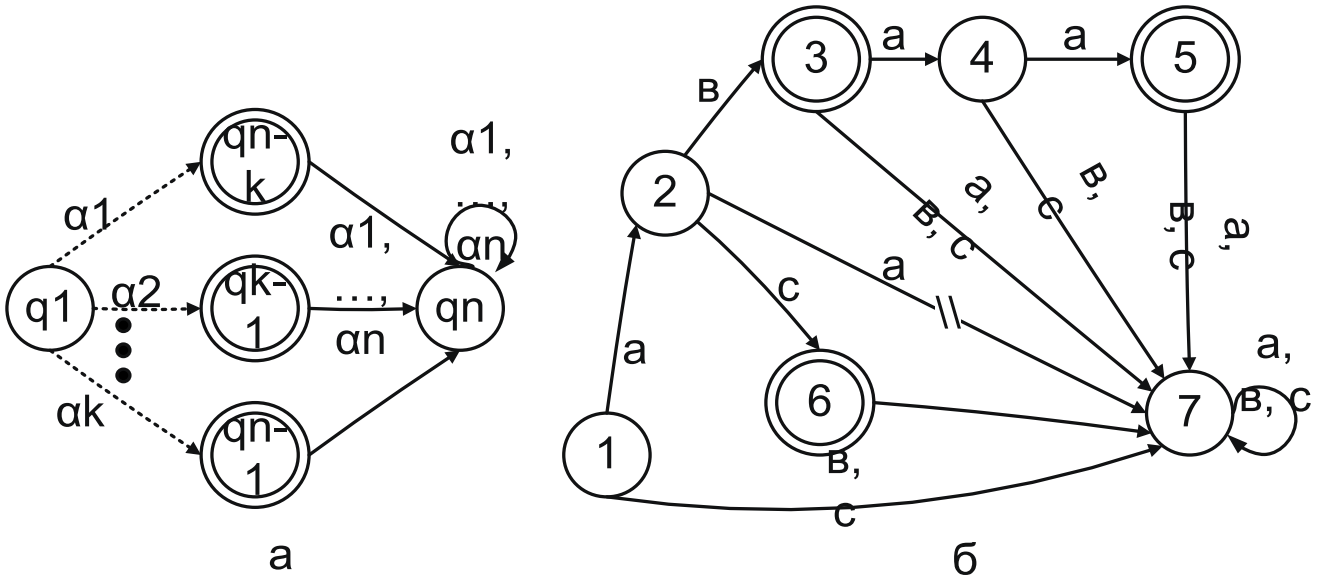


Рис. 3.6. Автомат Мілі і автомат Мура для скінченної множини слів $E = \{a_1,...,a_k\}$

Для конкретних множин ця ідея модифікується у зв'язку з тим, що слова можуть мати загальні початки (тоді початки відповідних шляхів потрібно об'єднати, щоб не порушити умову автоматності) або просто втримувалися один в одному (тоді з одного заключного стану є шлях в інший заключний стан). Приклад автомата для $E = \{ab, ac, abaa\}$ із заключними станами $F = \{3,5,6\}$ показано на рис. 3.6, б.

а) в автоматі, що являє собою скінченну множину слів, шлях з початкового стану в будь-який заключний стан не може містити циклів або втримуватися в циклі, оскільки тоді була б нескінченна множина шляхів з початкового стану в F, що свідчить про те, що відповідна подія була б нескінченною. Тому такий автомат не може бути сильно-зв'язаним, він є пристроєм одноразової дії;

б) автономні автомати представляють події в однобуквеному алфавіті; слова в таких подіях відрізняються тільки довжиною. Наприклад, автомат з початковим станом 1 і $F = \{7\}$ (виходи опускаємо) являє собою нескінченну подію, що складається із усіх слів, довжини яких при діленні на 4 дають в остачі 3. Якщо ж покласти $F = \{2\}$, то цей автомат є порожньою подією;

в) автомат, граф якого показано на рис. 3.7 ($F = \{1\}$), є нескінченною множиною $\{e, aba, abaaba, ..., \{aba\}\}$.

Події – це множини скінченних слів. Однак можна говорити, що автомат розпізнає нескінченну послідовність букв $\alpha = a_{i1}, a_{i1}, a_{i1}...,$, якщо він є множиною $E = \{a_{i1}, a_{i1}, a_{i1}, ...\}$, що складається з усіх початкових відрізків послідовності α.

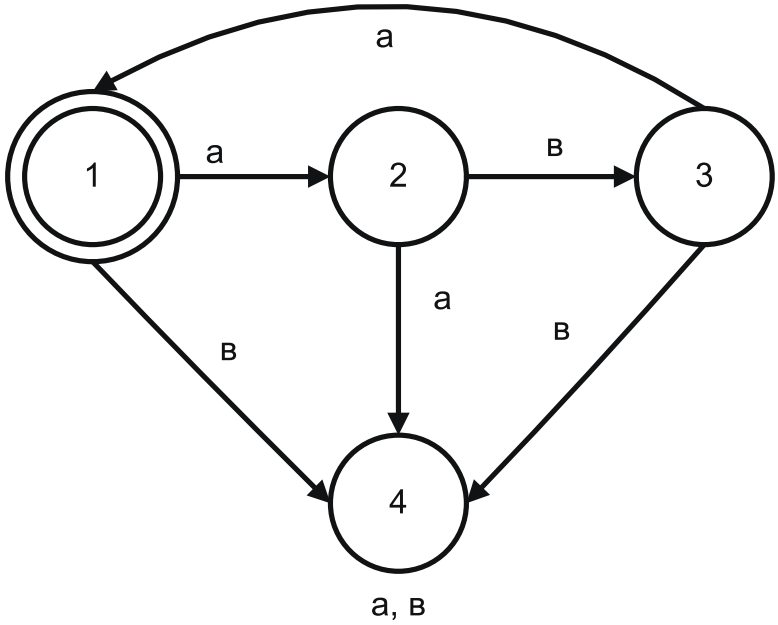


Рис. 3.7. Автомат нескінченної множини *{e,aba,abaaba,...,{aba}}*

КОНТРОЛЬНІ ЗАПИТАННЯ Й ЗАВДАННЯ ДЛЯ САМОКОНТРОЛЮ

1) Загальне визначення скінченного автомата.
2) У чому полягає основна суть скінченного автомата?
3) Інтерпретація скінченного автомата.
4) Що називається недетермінованим скінченним автоматом?
5) Скінченний автомат з двома стрічками.
6) Автомат Мілі.
7) Автомат Мура.
8) Як перетворити автомат Мілі в автомат Мура?
9) Який зв'язок між моделями Мілі й Мура?

ТЕМИ ДЛЯ САМОСТІЙНОЇ РОБОТИ

1) Лінійно обмежені автомати.
2) Зв'язок між лінійно обмеженими автоматами й контекстно-залежними граматиками.
3) Теорема Курода.
4) Контекстно-залежні мови.
5) Автомат з магазинною пам'яттю.
6) Характеризация контекстно-вільних мов.
7) Автомати з магазинною пам'яттю з однобуквеними переходами.
8) Теорема Хейса - Шютценбержа.
9) Приклади недетермінованого автомата з магазинною пам'яттю.
10) Алгоритмічні проблеми.
11) Алгоритмічно нерозв'язні проблеми.
12) Алгоритмічно розв'язні проблеми

Бібліографічний список

1. Аляев, Ю.А. Дискретная математика и математическая логика [Текст] : учебник / Ю.А. Аляев, С.Ф. Тюрин. - М. : Финансы и статистика, 2006. - 368 с.

2. Гуц, А.К. Математическая логика и теория алгоритмов [Текст] : учеб. пособие / А.К. Гуц. - Омск : Наследие. Диалог-Сибирь, 2003. - 108 с.

3. Игошин, В.И. Математическая логика и теория алгоритмов [Текст] : учеб. пособие / В.И. Игошин. - М. : Центр «Академия», 2008. - 448 с.

4. Колмогоров, А.Н. К определению алгоритма [Текст] : / А.Н. Колмогоров, В.А. Успенский // Успехи математических наук. - 1958. - Т. 13. - № 4(82). - С. 3 - 28.

5. Формальная грамматика [Электронный ресурс]. - Режим доступа к документу: http://ru.wikipedia.org/wiki/Формальная_грамматика. Заголовок с экрана.

6. Хопкрофт, Д. Контекстно-свободные грамматики и языки [Текст] / Джон Хопкрофт, Раджив Мотвани, Джеффри Ульман // Введение в теорию автоматов, языков и вычислений. Introduction to Automata Theory, Languages, and Computation. - М. : Вильямс, 2002. – 528 с.

Зміст

Навчальне видання

Шостак Ігор Володимирович
Груздо Ірина Володимирівна
Данова Марія Олександрівна
Бутенко Юлія Іванівна

ТЕОРІЯ АЛГОРИТМІВ І ОБЧИСЛЮВАЛЬНИХ ПРОЦЕСІВ

Редактор С.П. Гевло

Зв. план, 2013
Підписано до друку 20.08.2013
Формат 60×84 1/16. Папір офс. № 2. Офс. друк
Ум. друк. арк. 4. Обл.-вид. арк. 4,76. Наклад 100 пр.
Замовлення 270. Ціна вільна

Національний аерокосмічний університет ім. М.Є. Жуковського
"Харківський авіаційний інститут"
61070, Харків-70, вул. Чкалова, 17
http://www.khai.edu
Видавничий центр «ХАІ»
izdat@khai.edu
61070, Харків-70, вул. Чкалова, 17

Свідоцтво про внесення суб'єктів видавничої справи
до Державного реєстру видавців, виготовлювачів і розповсюджувачів
видавничої продукції сер. ДК № 391 від 30.03.2001