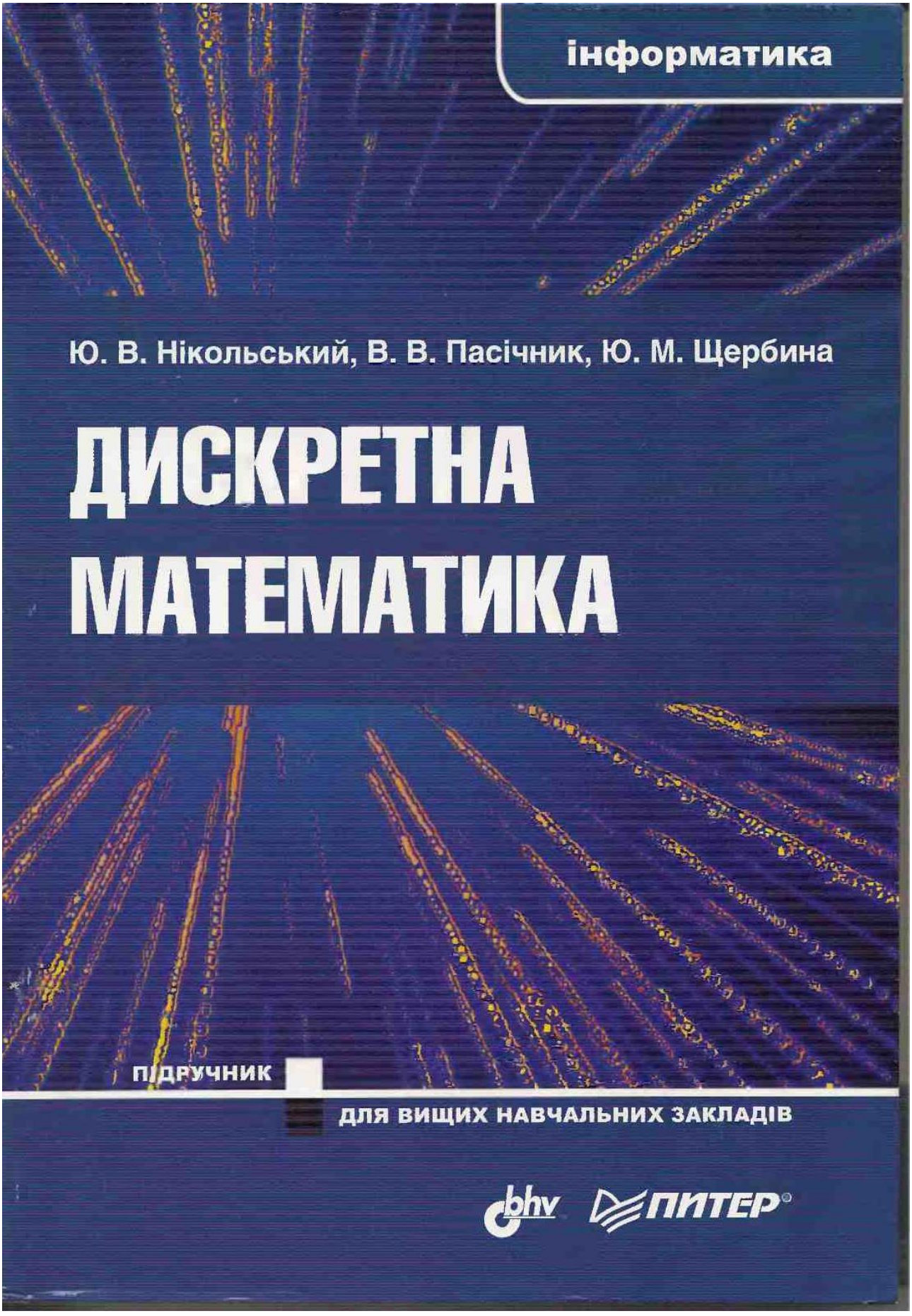
інформатика
Ю. В. Нікольський, В. В. Пасічник, Ю. М. Щербина
ДИСКРЕТНА МАТЕМАТИКА
ПІДРУЧНИК
ДЛЯ ВИЩИХ НАВЧАЛЬНИХ ЗАКЛАДІВ
bhv
ПИТЕР

О. В. НІКОЛЬСЬКИЙ, В. В. ПАСІЧНИК, Ю. М. ЩЕРБИНА

ДИСКРЕТНА МАТЕМАТИКА

Затверджено Міністерством освіти і науки України як підручник для студентів вищих навчальних закладів, які навчаються за напрямами «Комп'ютерні науки», «Комп'ютеризовані системи, автоматика та управління», «Комп'ютерна інженерія», «Прикладна математика»

Серія «ІНФОРМАТИКА»
За загальною редакцією академіка НАН України
М. З. Згуровського

Київ
Видавнича група BHV
2007

Рецензенти: *Приймак М. В.*, завідувач кафедри комп'ютерних наук Тернопільського державного технічного університету ім. Івана Пулюя, доктор технічних наук, професор.
Цегелик Г. Г., завідувач кафедри математичного моделювання соціально-економічних процесів Львівського національного університету ім. Івана Франка, доктор фізико-математичних наук, професор.

Гриф надано Міністерством освіти і науки України,
лист № 14/18.2-333 від 13.02.2006 р.

Нікольський Ю. В., Пасічник В. В., Щербина Ю. М.

Н64 Дискретна математика. — К.: Видавнича група BHV, 2007. — 368 с.: іл.
ISBN 966-552-201-9

У підручнику в логічній послідовності викладено основні поняття та методи дискретної математики. Окрім таких розділів, як теорія множин і математична логіка, теорія графів, основи теорії кодування, теорія булевих функцій, теорія алгоритмів та формальних мов, які традиційно входять до базового курсу дисципліни, розглянуто також основи теорії складності обчислень та деякі застосування дискретної математики у штучному інтелекті.

За змістом та обсягом підручник відповідає навчальним планам дисципліни «Дискретна математика» для студентів базових напрямів «Комп'ютерні науки», «Комп'ютеризовані системи, автоматика та управління», «Комп'ютерна інженерія» та «Прикладна математика».

ББК 22.176я73

Серія підручників «Інформатика» виходить у світ за підтримки видавництв «Издательский дом „Питер"», м. Санкт-Петербург, та «Освітня книга», м. Київ.

ISBN 966-552-201-9

Зміст

Розділ 9

Основи теорії алгоритмів

- Основні вимоги до алгоритмів
- Машини Тьюрінга
- Обчислення числових функцій на машинах Тьюрінга
- Теза Тьюрінга. Приклади алгоритмічно нерозв'язних проблем
- Рекурсивні функції
- Теза Чорча. Зв'язок рекурсивних функцій із машинами Тьюрінга

Поняття алгоритму — одне з найголовніших у математиці. Глибоке розуміння його потрібне, по-перше, для розробки конкретних алгоритмів, особливо коли потім планують програмувати їх. По-друге, шоб орієнтуватись у великій кількості алгоритмів, треба вміти порівнювати різні алгоритми розв'язування одних і тих самих задач, причому не тільки за якістю розв'язку, але й за характеристиками самих алгоритмів (кількістю операцій, потрібним обсягом пам'яті тощо). Таке порівняння неможливе без уведення точної мови для пояснення цих понять. По-третє, може виникнути потреба довести, що для розв'язання певного класу задач алгоритму взагалі не існує. Отже, слід розглядати алгоритми як об'єкти точного дослідження.

9.1. Основні вимоги до алгоритмів

Строге дослідження основних понять теорії алгоритмів розпочнемо в наступному підрозділі, а тепер неформально розглянемо деякі головні принципи, на яких будують алгоритми, і виявимо, що саме слід уточнити в понятті алгоритму [21].

1. Будь-який алгоритм застосовують до початкових даних, і він видає результати. У звичних технічних термінах це означає, що алгоритм має входи й виходи. Отже, алгоритм застосовують для розв'язування цілого класу задач із різними початковими даними (цю властивість називають *масовістю*). Крім того, під час роботи алгоритму з'являються проміжні результати, використовувані в подальшому. Звідси випливає, що, кожний алгоритм обробляє *дані* — вхідні, проміжні та вихідні. Оскільки ми збираємось уточнити поняття алгоритму, потрібно уточнити й поняття даних, тобто зазначити, яким вимогам мають задовольняти об'єкти, щоб алгоритми могли з ними працювати.
2. Для розміщення даних потрібна пам'ять. Її зазвичай уважають однорідною й дискретною, тобто такою, що складається з однакових комірок, причому кож-

на комірка може містити один символ алфавіту даних. Отже, одиниці виміру обсягу даних і пам'яті узгоджені, при цьому пам'ять може бути нескінченною. Питання про те, чи потрібна одна пам'ять, чи декілька й, зокрема, чи потрібна окрема пам'ять для кожного з трьох типів даних, вирішують по-різному.

3. Алгоритм складається з окремих *елементарних кроків* (або *дій*), причому множина різних кроків, з яких складено алгоритм, скінченна. Типовий приклад множини елементарних дій — система команд процесора. Зазвичай на елементарному кроці обробляється фіксована кількість символів, проте є команди, які працюють із полями пам'яті зі змінною довжиною.
4. Послідовність кроків алгоритму *детермінована*, тобто після кожного кроку зазначено, який крок робити далі, або виконується команда зупинки, після чого робота алгоритму вважається закінченою.
5. Алгоритм має бути *результативним*, тобто зупинятися після скінченної кількості кроків (залежно від початкових даних) із зазначенням того, що вважати результатом. Зокрема, алгоритм для обчислення функції $f(x)$ має зупинятися після скінченної кількості кроків для будь-якого x із області визначення функції f. У такому разі говорять, що алгоритм *збігається*. Проте перевірити результативність (збіжність) значно важче, ніж вимоги, викладені в пп. 1–4. На відміну від них збіжність переважно неможливо виявити простим переглядом опису алгоритму. Загального ж методу перевірки збіжності, придатного для довільного алгоритму A й довільних початкових даних α, узагалі не існує (див. підрозділ 9.4).

Потрібно розрізняти:

- опис алгоритму (інструкцію чи програму);
- механізм реалізації алгоритму (наприклад, персональний комп'ютер), який містить засоби запуску, зупинки, реалізації елементарних кроків, видачі результатів і забезпечення детермінованості, тобто керування перебігом обчислення;
- процес реалізації алгоритму, тобто послідовність кроків, яка буде породжена в разі застосування алгоритму до конкретних даних.

Уважатимемо, що опис алгоритму й механізм його реалізації скінченні (пам'ять, як ми вже зазначали, може бути нескінченною, але вона не входить у механізм). Вимога скінченності процесу реалізації збігається з вимогою результативності (див. п. 5).

Ми сформулювали головні вимоги до алгоритмів. Прості поняття, використані в них, інтуїтивні. Тому в теорії алгоритмів застосовують інший підхід: вибирають скінченний набір основних об'єктів, які оголошують елементарними, і скінченний набір способів побудови з них нових об'єктів. Уточненням поняття „дані" вважатимемо множини слів (ланцюжків) у скінченних алфавітах. Для уточнення детермінізму використовуватимемо опис механізму реалізації алгоритму. Крім того, потрібно зафіксувати набір елементарних кроків і домовитися про організацію пам'яті. Коли це буде зроблено, отримаємо конкретну алгоритмічну модель.

Алгоритмічні моделі, які ми розглянемо в цьому розділі, можна вважати формалізацією поняття „алгоритм". Це означає, що вони мають бути універсальними, тобто за їх допомогою можна описати будь-які алгоритми. Тому може виникнути природне заперечення проти запропонованого підходу: чи не призведе

вибір конкретних засобів до втрати загальності формалізації? Маючи на увазі головні цілі, поставлені під час розробки теорії алгоритмів, — універсальність і пов'язану з нею можливість говорити в рамках якоїсь моделі про властивості алгоритмів узагалі, — це заперечення можна зняти так. По-перше, доводять звідність одних моделей до інших, тобто те, що будь-який алгоритм, описаний засобами однієї моделі, можна описати й засобами іншої. По-друге, завдяки взаємній звідності моделей у теорії алгоритмів удалося виробити інваріантну щодо моделей систему понять, яка дає змогу говорити про властивості алгоритмів незалежно від того, яку формалізацію алгоритму вибрано. Ця система ґрунтується на понятті *обчислюваної функції*, тобто такої, для обчислення якої існує алгоритм.

Однак, хоча загальність формалізації в конкретній моделі не втрачається, унаслідок різного вибору початкових засобів виникають різні моделі. Можна виділити три основні типи універсальних алгоритмічних моделей, які різняться початковими евристичними міркуваннями відносно того, що таке алгоритм.

У першому типі поняття алгоритму пов'язане з найтрадиційнішими поняттями математики — обчисленнями та числовими функціями (числовою називають функцію, значення якої та значення її аргументів — невід'ємні цілі числа). Найпопулярніша модель цього типу — *рекурсивні функції* — історично перша формалізація поняття алгоритму.

Другий тип ґрунтується на уявленні про алгоритм як детермінований пристрій, здатний виконувати в кожний окремий момент лише дуже примітивні операції. У такому разі не залишається сумнівів у однозначності алгоритму й елементарності його кроків. Окрім того, евристика цих моделей близька до комп'ютерів. Основна теоретична модель цього типу (створена в 30-х роках XX століття — раніше комп'ютерів) — *машина Тьюрінга.*

Нарешті, третій тип алгоритмічних моделей — це перетворення слів у довільних алфавітах; у цих моделях елементарні операції — це підстановки, тобто заміна частини слова (підслова) іншим словом. Приклади моделей цього типу — *канонічні системи Поста* й *нормальні алгорифми Маркова.*

Розглянемо алгоритмічні моделі двох перших типів; почнемо з машин Тьюрінга.

9.2. Машини Тьюрінга

Машина Тьюрінга — це математична модель пристрою, який породжує обчислювальні процеси. Її використовують для теоретичного уточнення поняття алгоритму та його дослідження. Названо цю модель ім'ям англійського математика А. Тьюрінга (A. Turing), який запровадив її 1936 р. У кожній машині Тьюрінга є три частини (рис. 9.1).

1. Стрічка, поділена на комірки.
2. Пристрій керування (ПК).
3. Головка читання–запису (Г).

Із кожною машиною Тьюрінга пов'язано два скінченні алфавіти: *алфавіт зовнішніх символів A* й *алфавіт внутрішніх станів* $Q = \{q_0, q_1, ..., q_k\}$. Із різними машинами Тьюрінга можна пов'язувати різні алфавіти.

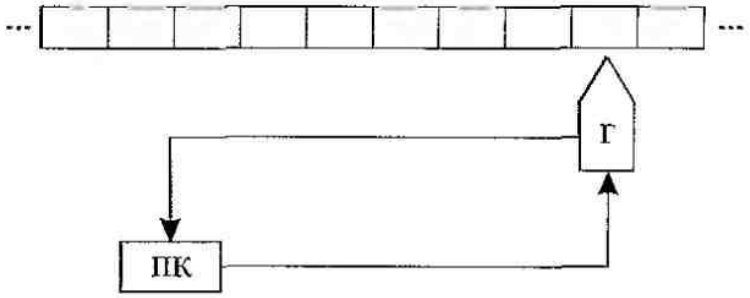

Рис. 9.1

Алфавіт зовнішніх символів A часто називають *зовнішнім*, а його елементи — *буквами*. Один символ з алфавіту A називають *порожнім*, зазвичай його позначають Λ. Усі інші букви з алфавіту A, крім Λ, називають *непорожніми*. Комірку стрічки, у якій записано букву Λ, називають *порожньою*.

Машина Тьюрінга працює в часі, який уважають дискретним; його моменти занумеровано: 1, 2, 3, ... У кожний момент стрічка містить скінченну кількість комірок. Головка пересувається вздовж стрічки; у кожний момент вона перебуває над певною коміркою стрічки. У такому разі говорять, що головка *зчитує* букву, записану в цій комірці. У наступний момент головка залишається над цією самою коміркою (що позначають Н), або пересувається на одну комірку вправо (що позначають П), або пересувається на одну комірку вліво (що позначають Л). Якщо в даний момент t головка перебуває над крайньою коміркою та зсувається на комірку, якої немає, то автоматично прибудовується нова порожня (тобто з порожньою буквою Λ) комірка, над якою головка перебуватиме в момент $t + 1$. Отже, стрічка *потенційно нескінченна в обидва боки*, тобто до неї завжди можна додати нові комірки як зліва, так і справа.

Алфавіт внутрішніх станів $Q = \{q_0, q_1, ..., q_k\}$ — це *внутрішня пам'ять*. На відміну від нескінченної *зовнішньої пам'яті* на стрічці, вона скінченна. Елемент q_0 називають *заключним внутрішнім станом*, а елемент q_1 — *початковим внутрішнім станом*. Пересування головки вздовж стрічки залежить від зчитаної букви та внутрішнього стану машини.

У кожний момент t залежно від букви, зчитуваної зі стрічки, і внутрішнього стану машини пристрій керування виконує такі дії.

1. Змінює букву a_i, зчитувану в момент t зі стрічки, на нову букву a_j (зокрема, може бути $a_j = a_i$).
2. Пересуває головку в одному з напрямків Н, П, Л.
3. Змінює внутрішній стан машини q_i в момент t на новий стан q_p у якому машина перебуватиме в момент $t + 1$ (зокрема, може бути $q_p = q_i$).

Таку дію пристрою керування називають *командою* й записують

$$q_i a_i \rightarrow a_j D q_p$$

де q_i — внутрішній стан машини в даний момент; a_i — буква на стрічці, зчитувана в цей момент; a_j — буква, на яку змінюється буква a_i (може бути $a_j = a_i$); D — Н, П чи Л — напрямок пересування головки; q_p — внутрішній стан машини в наступний момент часу (може бути $q_p = q_i$).

Вирази $q_i a_i$ й $a_j D q_j$ називають відповідно лівою та правою частинами цієї команди. У лівій частині немає жодної команди q_0. Усіх команд, у яких ліві частини попарно різні, скінченна кількість (бо множини $Q \setminus \{q_0\}$ й A скінченні); їх сукупність називають *програмою машини*. Жодні дві команди не можуть мати однакові ліві частини.

Виконання однієї команди називають *кроком*. Обчислення (або робота) машини Тьюрінга — це послідовність кроків одного за другим без пропусків, починаючи з першого. Роботу машини Тьюрінга повністю визначено, якщо в перший (тобто початковий) момент задано.

1. Слово на стрічці, тобто послідовність букв, записаних у її комірках (слово одержують читанням цих букв у комірках стрічки зліва направо).
2. Положення головки Г.
3. Внутрішній стан машини.

Сукупність цих трьох умов (у даний момент t) називають *конфігурацією* (у даний момент t). Зазвичай у початковий момент внутрішній стан машини — q_1, а головка перебуває над першою зліва коміркою стрічки.

Отже, у початковий момент конфігурація така: на стрічці, що складається з якоїсь кількості комірок (не менше однієї), у кожній комірці записано одну з букв зовнішнього алфавіту A, головка над першою зліва коміркою стрічки, внутрішній стан машини — q_1. Наприклад, якщо в початковий момент на стрічці записано слово $a_1 \Lambda a_2 a_1 a_1$ то початкова конфігурація така (рис. 9.2):

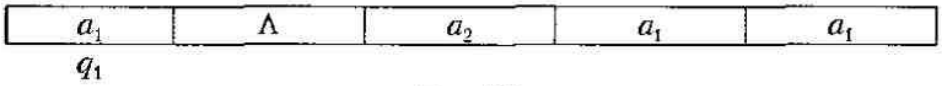

Рис. 9.2

(під коміркою, над якою перебуває головка, зазначено внутрішній стан машини). Отже, програма й початкова конфігурація повністю задають роботу машини над словом, яке є на стрічці в початковій конфігурації. Тому вважатимемо машину Тьюрінга заданою, якщо відома її програма.

Якщо в роботі машини Тьюрінга в момент t виконається команда, права частина котрої мітить символ q_0, то в такий момент роботу машини вважають завершеною; тоді говорять, що машина *застосовна* до слова на стрічці в початковій конфігурації. Справді, символ q_0 не зустрічається в лівій частині жодної команди. тому його називають заключним станом. Результатом роботи машини в такому разі вважають слово, яке буде записано на стрічці в *заключній конфігурації*, тобто в конфігурації з внутрішнім станом q_0.

Якщо ж у роботі машини в жодний момент не зустрінеться заключний стан. то процес обчислень буде нескінченним. У такому разі говорять, що машина *незастосовна* до слова на стрічці в початковій конфігурації.

Зазначимо, що $q_0 \in Q$ для кожної машини Тьюрінга.

Приклад 9.1. Побудуємо машину Тьюрінга T_1, котра застосовна до всіх слів у алфавіті $\{a, b\}$ та перетворює довільне слово $x_1 x_2 \ldots x_n$, де $x_i \in \{a, b\}$, $i = 1, \ldots, n$ на слово $x_2 \ldots x_n x_1$.

Як зовнішній алфавіт машини візьмемо множину $A = \{\Lambda, a, b\}$; команди разом із коментарями запишемо у два стовпчики. Для запам'ятовування букв алфавіту стрічки потрібно перейти в новий стан.

Команда	Коментар
$q_1a \to \Lambda\Pi q_2$	Перша буква a запам'ятовується переходом у стан q_2 та стирається
$q_1b \to \Lambda\Pi q_3$	Перша буква b запам'ятовується переходом у стан q_3 та стирається
$q_2a \to a\Pi q_2$ $q_2b \to b\Pi q_2$ $q_2\Lambda \to a\text{H}q_0$	Проходження головки через слово $x_2 \ldots x_n$ і запис у його кінці букви $x_1 = a$, зупинка
$q_3a \to a\Pi q_3$ $q_3b \to b\Pi q_3$ $q_3\Lambda \to b\text{H}q_0$	Проходження головки через слово $x_2 \ldots x_n$ і запис у його кінці букви $x_1 = b$, зупинка

Ці команди можна записати у вигляді таблиці, яку називають *функціональною схемою Тьюрінга* (табл. 9.1).

Таблиця 9.1

Стан	Λ	a	b
q_1	–	$\Lambda\Pi q_2$	$\Lambda\Pi q_3$
q_2	$a\text{H}q_0$	$a\Pi q_2$	$b\Pi q_2$
q_3	$b\text{H}q_0$	$a\Pi q_3$	$b\Pi q_3$

У таблиці прочерком позначено неістотну команду, тобто таку, яка не зустрінеться під час роботи цієї машини.

Розглянемо роботу машини T_1 над словом abb. Конфігурації, які при цьому виникають, зображено на рис. 9.3.

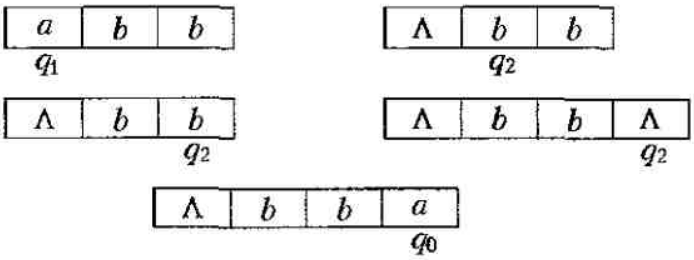

Рис. 9.3

Приклад 9.2. Побудуємо машину Тьюрінга T_2, яка застосовна до всіх слів у алфавіті $\{a, b\}$ та знімає копію слова на стрічці, тобто перетворює слово $x_1 \ldots x_n$ у початковій конфігурації на слово $x_1 \ldots x_n \Lambda x_1 \ldots x_n$ у заключній конфігурації.

Цю машину можна побудувати, наприклад, так. Нехай задано якесь слово в алфавіті $\{a, b\}$, що складається з n букв ($n \geqslant 1$). Робота машини містить n циклів. Для $i \leqslant n$ на початок i-го циклу конфігурацію зображено на рис. 9.4. Отже, скопійовано $i-1$ букв слова, і головка зчитує букву x_i. Черговий цикл виконуватиметься в три етапи.

1. Запам'ятовування букви x_i (при цьому вона позначається, тобто фактично заміюється іншою буквою) та пересування головки вправо.
2. Пошук правої крайньої порожньої комірки, у яку записується x_i.
3. Повернення головки вліво до позначеної букви, стирання мітки (тобто відновлення букви x_i) та пересування головки на одну клітку вправо.

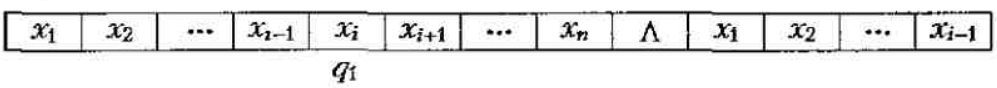

Рис. 9.4

Перший етап реалізують такі команди.

Команда	Коментар
$q_1a \to a'\Pi q_2$	a' — позначена буква a, q_2 — стан запам'ятовування букви a
$q_1b \to b'\Pi q_3$	b' — позначена буква b, q_3 — стан запам'ятовування букви b

Другий етап.

Команда	Коментар
$q_2a \to a\Pi q_2$ $q_2b \to b\Pi q_2$	Проходження залишку слова в стані q_2
$q_3a \to a\Pi q_3$ $q_3b \to b\Pi q_3$	Проходження залишку слова в стані q_3
$q_2\Lambda \to \Lambda\Pi q_4$ $q_3\Lambda \to \Lambda\Pi q_5$	Проходження головки через порожню букву Λ між словами; q_4 — копіюватиметься буква a; q_5 — копіюватиметься буква b
$q_4a \to a\Pi q_4$ $q_4b \to b\Pi q_4$ $q_5a \to a\Pi q_5$ $q_5b \to b\Pi q_5$	Проходження початкової частини копії слова
$q_4\Lambda \to a\Pi q_6$ $q_5\Lambda \to b\Pi q_6$	Запис копійованої букви

Третій етап.

Команда	Коментар
$q_6a \to a\Pi q_6$ $q_6b \to b\Pi q_6$ $q_6\Lambda \to \Lambda\Pi q_6$ $q_6a' \to a\Pi q_1$ $q_6b' \to b\Pi q_1$	Рух головки вліво до позначеної букви, стирання мітки, пересування на одну клітку вправо з переходом у початковий стан q_1. Машина готова до виконання чергового циклу копіювання букви
$q_1\Lambda \to \Lambda\Pi q_0$	Команда зупинки, копію слова побудовано

Отже, зовнішній алфавіт машини T_2 — множина $A = \{\Lambda, a, b, a', b'\}$, множина внутрішніх станів $Q = \{q_0, q_1, ..., q_6\}$. Нарешті, запишемо команди у вигляді функціональної схеми Тьюрінга (табл. 9.2).

Таблиця 9.2

Стан	Λ	a	b	a'	b'
q_1	$\Lambda\Pi q_0$	$a'\Pi q_2$	$b'\Pi q_3$	–	–
q_2	$\Lambda\Pi q_4$	$a\Pi q_2$	$b\Pi q_2$	–	–
q_3	$\Lambda\Pi q_5$	$a\Pi q_3$	$b\Pi q_3$	–	–
q_4	$a\Pi q_6$	$a\Pi q_4$	$b\Pi q_4$	–	–
q_5	$b\Pi q_6$	$a\Pi q_5$	$b\Pi q_5$	–	–
q_6	$\Lambda\Pi q_6$	$a\Pi q_6$	$b\Pi q_6$	$a\Pi q_1$	$b\Pi q_1$

Алфавіт зовнішніх символів A в цьому прикладі крім порожньої букви Λ й букв a та b, у яких записується слово для копіювання, містить іще додаткові букви a', b', використовувані в проміжних обчисленнях.

9.3. Обчислення числових функцій на машинах Тьюрінга

Числовою називають функцію $f(x_1, ..., x_n)$, значення якої та значення її аргументів — невід'ємні цілі числа. Розглянемо часткові числові функції, визначені, узагалі кажучи, не для всіх значень аргументів.

Для обчислення числових функцій на машинах Тьюрінга застосовують спеціальне кодування чисел. Наприклад [30], невід'ємне ціле число m можна задати набором з $(m + 1)$ одиниць, який позначатимемо 1^{m+1}: 0 — як 1, 1 — як 11, 2 — як 111 тощо.

Числову функцію $f(x_1, ..., x_n)$ називають *обчислюваною за Тьюрінгом*, якщо існує така машина Тьюрінга T, що для довільних цілих невід'ємних чисел $m_1, m_2, ..., m_n$, якщо $f(m_1, m_2, ..., m_n) = m$, то машина T застосовна до слова $1^{m_1+1}\Lambda 1^{m_2+1}\Lambda ... \Lambda 1^{m_n+1}$ і в заключній конфігурації на якомусь відрізку стрічки буде записано слово 1^{m+1}, а решта комірок (якщо такі є) виявляться порожніми. Якщо ж значення $f(m_1, m_2, ..., m_n)$ не визначено, то машина T незастосовна до слова $1^{m_1+1}\Lambda 1^{m_2+1}\Lambda ... \Lambda 1^{m_n+1}$.

Приклад 9.3. Побудуємо машину Тьюрінга, яка обчислює числову функцію $s(x) = x + 1$. Зовнішній алфавіт $A = \{\Lambda, 1\}$, множина станів $Q = \{q_0, q_1\}$, а команди можна задати так: $q_1 1 \to 1\Pi q_1$, $q_1\Lambda \to 1Hq_0$. Конфігурації, що виникають у разі обчислення $s(1)$, зображено на рис. 9.5.

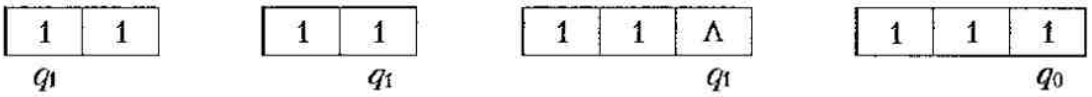

Рис. 9.5

Приклад 9.4. Побудуємо машину Тьюрінга, яка обчислює числову функцію $f(x, y) = x + y$. Зовнішній алфавіт $A = \{\Lambda, 1\}$, множина станів $Q = \{q_0, q_1, q_2, q_3, q_4\}$. У процесі роботи машини головка рухається вправо, і символ Λ між аргументами замінюється на 1. Далі продовжується рух праворуч до першої комірки із символом Λ, після чого головка починає рухатись уліво та стирає дві останні 1 поспіль. Команди можна задати так: $q_1 1 \to 1\Pi q_1$, $q_1\Lambda \to 1\Pi q_2$, $q_2 1 \to 1\Pi q_2$, $q_2\Lambda \to \Lambda\Pi q_3$, $q_3 1 \to \Lambda\Pi q_4$, $q_1 1 \to \Lambda H q_0$. Конфігурації, що виникають під час обчислення $f(1, 2)$, зображено на рис. 9.6.

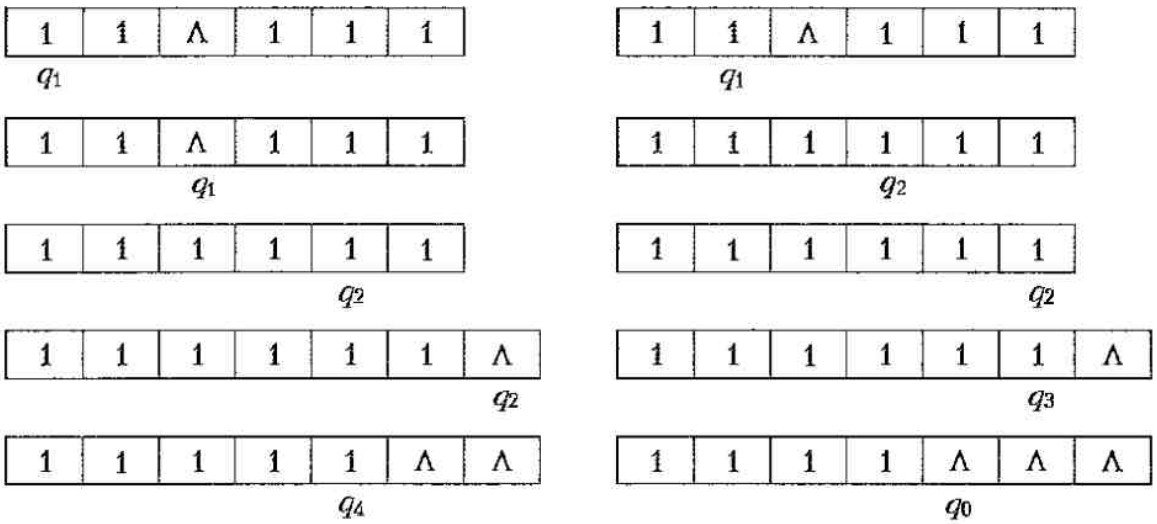

Рис. 9.6

9.4. Теза Тьюрінга. Приклади алгоритмічно нерозв'язних проблем

До цього часу нам удавалося для всіх процедур, які „претендують" на алгоритмічність, тобто *ефективних*, будувати машини Тьюрінга, що їх реалізують. Чи правильно це загалом? Ствердна відповідь на це питання міститься в *тезі Тьюрінга*, яку формулюють так: *будь-який алгоритм можна реалізувати машиною Тьюрінга.*

Довести тезу Тьюрінга неможливо, бо саме поняття алгоритму (ефективної процедури) інтуїтивне (неточне). Це не теорема й не аксіома математичної теорії, а твердження, яке пов'язує теорію з тими об'єктами, для яких її побудовано. За своїм характером теза Тьюрінга нагадує гіпотези фізики про адекватність математичних моделей фізичним явищам і процесам. Підтверджує тезу Тьюрінга, по-перше, математична практика, по-друге — та обставина, що опис алгоритму в термінах будь-якої іншої відомої алгоритмічної моделі можна звести до його опису у вигляді машини Тьюрінга.

Теза Тьюрінга дає змогу, по-перше, замінити неточні (інтуїтивні) твердження про існування ефективних процедур (алгоритмів) точними твердженнями про існування машин Тьюрінга, а, по-друге, тлумачити твердження про неіснування машин Тьюрінга як твердження про неіснування алгоритму.

Розглянемо тепер два приклади алгоритмічно нерозв'язних проблем [21, 30].

Проблема самозастосовності. Закодуємо всі машини Тьюрінга символами алфавіту {*, 1} так, щоб за системою команд машини Тьюрінга можна було ефективно побудувати код машини та, навпаки, за кодом можна було ефективно записувати її програму. Це можна зробити, наприклад, так. Нехай T — будь-яка машина Тьюрінга, зовнішній алфавіт якої — $\{\Lambda, a_1, a_2, ..., a_m\}$, а множина внутрішніх станів — $\{q_0, q_1, ..., q_k\}$. Занумеруємо всі символи, що є в командах (окрім $\rightarrow$), числами 0, 1, 2, ... так, як це показано в табл. 9.3.

Таблиця 9.3

Символ	Н	Л	П	Λ	a_1	...	a_m	q_0	q_1	...	q_k
Номер	0	1	2	3	4	...	$m+3$	$m+4$	$m+5$	...	$m+4+k$

Число i подаватимемо набором з $i + 1$ одиниць. Якщо b — один із символів Н, Л, П, Λ, a_1, ... , a_m, q_0, q_1, ... , q_k, то як $N(b)$ позначимо набір з одиниць, кількість яких відповідає номеру символу b. Наприклад, N(Н) = 1, N(Л) = 11, N(П) = 111, $N(\Lambda) = 1111$, $N(a_1) = 11111$, $N(a_2) = 111111$. Команді $q_i a_i \rightarrow a_j D q_j$ поставимо у відповідність слово

$$C(q_i a_i) = N(q_i) * N(a_i) * N(a_j) * N(D) * N(q_j),$$

яке називають *кодом* цієї *команди*. *Кодом машини* T називають слово

$$W(T) = **C(q_1\Lambda)**C(q_1a_1)**...**C(q_1a_m)**C(q_2\Lambda)**C(q_2a_1)**...$$
$$...**C(q_2a_m)**C(q_3\Lambda)**...**C(q_k\Lambda)**...**C(q_ka_m)**$$

(тут у визначеному порядку записано коди всіх команд, відокремлені один від другого двома зірочками).

Розглядатимемо машини Тьюрінга, зовнішній алфавіт яких містить символи * й 1. Нехай T — одна з таких машин. Якщо машина T застосовна до слова $W(T)$, тобто до власного коду, то її називають *самозастосовною*, а ні, то *несамозастосовною*. Кожна машина розглянутого типу або самозастосовна, або несамозастосовна.

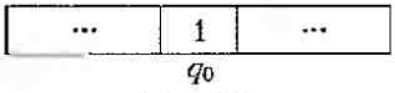

Рис. 9.7

Проблема самозастосовності полягає в тому, що потрібно навести алгоритм для визначення того, чи дана машина Тьюрінга самозастосовна, чи ні. Згідно з тезою Тьюрінга цю задачу можна сформулювати як задачу про побудову машини Тьюрінга, котра застосовна до кодів усіх машин, і заключні конфігурації в разі роботи над кодами самозастосовних і несамозастосовних машин відрізняються. Наприклад, вимагатимемо, щоб у разі роботи над кодом самозастосовної машини заключна конфігурація мала такий вигляд, як на рис. 9.7, а в разі роботи над кодом несамозастосовної машини — як на рис. 9.8. На цих рисунках поза зчитуваним символом можуть бути будь-які символи зовнішнього алфавіту.

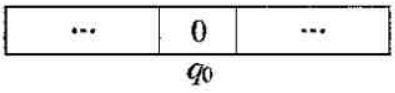

Рис. 9.8

ТЕОРЕМА 9.1. Проблема самозастосовності алгоритмічно нерозв'язна, тобто не існує машини Тьюрінга, яка розв'язує в наведеному вище розумінні цю проблему.

Доведення. Припустимо протилежне. Нехай існує машина L, яка розв'язує проблему самозастосовності. За нею побудуємо машину T, що має такі властивості.

1. Машина T застосовна до кодів несамозастосовних машин.
2. Машина T незастосовна до кодів самозастосовних машин.

Машину T будують так. Усі команди машини L оголошують командами машини T, але заключний стан q_0 машини L оголошують незаключним станом машини T. Будемо вважати заключним станом машини T новий стан q_0'; додамо до системи команд машини T ще такі дві команди:

$$q_0 0 \to 0 \mathrm{H} q_0' \quad \text{і} \quad q_0 1 \to 1 \mathrm{H} q_0.$$

Очевидно, що машина T має зазначені властивості 1 і 2, але вона чи самозастосовна, чи несамозастосовна. Нехай машина T самозастосовна, тобто вона застосовна до свого коду (коду самозастосовної машини), проте це суперечить властивості 2. Якщо ж машина T несамозастосовна, то вона незастосовна до свого коду (коду несамозастосовної машини). Але це суперечить властивості 1. Отже, машина T не може бути ні самозастосовною, ні несамозастосовною. Одержали суперечність. Оскільки машину T побудовано з машини L цілком конструктивно, то машини L не існує.

Проблема зупинки. Серед вимог, яким має задовольняти алгоритм, у підрозділі 9.1 було названо результативність. Найрадикальнішим її формулюванням була б вимога, щоб для будь-якого алгоритму A та даних α можна було визначити, чи приведе робота алгоритму A для початкових даних α до результату. Інакше кажучи, потрібно побудувати такий алгоритм B, що $B(A, \alpha) = 1$, якщо $A(\alpha)$ дає результат, і $B(A, \alpha) = 0$, якщо $A(\alpha)$ не дає результату.

Згідно з тезою Тьюрінга цю задачу можна сформулювати так. За машиною T та словом α розпізнати, чи зупиниться машина T, розпочавши роботу над словом α (тобто розпізнати, чи застосовна машина T до слова α). Отже, потрібно побудувати таку машину P, яка була б застосовна до всіх слів $W(T)\Lambda\alpha$, де T — довільна машина, α – довільне слово. Окрім того, заключні конфігурації машини P мають бути різними залежно від того, чи застосовна машина T до слова α, чи незастосовна.

ТЕОРЕМА 9.2. Проблема зупинки алгоритмічно нерозв'язна, тобто не існує машини Тьюрінга P, яка розв'язує в наведеному вище розумінні названу проблему.

Доведення. Нехай існує машина P, яка розв'язує проблему зупинки, а T — якась машина Тьюрінга. Тоді машина P має бути застосовна до всіх слів $W(T)\Lambda\alpha$, зокрема якщо $\alpha = W(T)$. Отже, машина P має бути застосовна до слова $W(T)\Lambda W(T)$. У разі роботи над словом $W(T)\Lambda W(T)$ вона зупиниться в конфігурації, зображеній на рис. 9.7, якщо машина T застосовна до слова $W(T)$ (тобто якщо машина T самозастосовна), і в конфігурації, зображеній на рис. 9.8, якщо машина T незастосовна до слова $W(T)$ (тобто якщо машина T несамозастосовна).

Нагадаємо, що машина, яка розв'язує проблему самозастосовності, починає працювати зі словом $W(T)$ на стрічці. Отже, щоб із машини P отримати машину, яка розв'язує проблему самозастосовності, використаємо машину T_2 з прикладу 9.2 (уважатимемо, що a — це *, b — 1). Машина T_2 за такої домовленості перетворює будь-яке слово β в алфавіті {*,1} на слово $\beta\Lambda\beta$. Машину Тьюрінга, яка розв'язує проблему самозастосовності, можна одержати з машин T_2 та P: спочатку працює машина T_2, а потім — машина P. Але це суперечить теоремі 9.1. Отже, машини P не існує.

Тлумачачи твердження, пов'язані з алгоритмічною нерозв'язністю, потрібно звернути увагу на те, що в них ідеться про те, що немає єдиного алгоритму, який розв'язує певну проблему. При цьому зовсім не виключено, що її можна розв'язати в часткових випадках, але за допомогою різних процедур для кожного з них. Тому нерозв'язність загальної проблеми зупинки зовсім не знімає потреби доводити збіжність пропонованих алгоритмів, а лише показує, що пошук таких доведень не можна повністю автоматизувати.

Нерозв'язність проблеми зупинки можна інтерпретувати як неіснування загального алгоритму для налагодження програм, котрий за текстом довільної програми й даними для неї міг би визначити, чи зациклиться програма на цих даних. Якщо врахувати зроблене перед цим зауваження, то така інтерпретація не суперечить тому емпіричному факту, що більшість програм удається налагоди-

ти, тобто виявити зациклення, знайти його причину й усунути її. При цьому вирішальну роль відіграють досвід і майстерність програміста.

9.5. Рекурсивні функції

Будь-який алгоритм однозначно ставить у відповідність початковим даним результат (якщо він визначений на них). Тому з кожним алгоритмом однозначно пов'язана функція, значення якої він обчислює. Чи правильне обернене твердження, тобто чи для будь-якої функції існує алгоритм, який обчислює її значення? Дослідження проблеми зупинки для машини Тьюрінга показує, що ні: не існує алгоритму обчислення значення предиката $P(T, \alpha)$, істинного якщо й лише якщо машина Тьюрінга T зупиняється на початкових даних α. Виникає запитання: для яких функцій існують алгоритми обчислення їх значень? Як описати такі алгоритмічні функції, тобто ефективно обчислювані функції?

Дослідження цих питань зумовило створення в 30-х роках XX століття теорії рекурсивних функцій. У ній, як і в теорії алгоритмів узагалі, прийнято конструктивний, фінітний підхід. Основна риса такого підходу полягає в тому, що вся множина об'єктів дослідження (у цьому разі – функцій) будується зі скінченної кількості вихідних об'єктів за допомогою простих операцій, ефективна виконуваність яких достатньо очевидна.

Розглядатимемо числові функції, тобто такі, що набувають значень — так само, як і їх аргументи — із множини $N_0 = \{0, 1, 2, ...\}$.

Назвемо найпростішими такі числові функції:

- *нуль-функцію*: $o(x) = 0$ для кожного x;
- *функцію наступності*: $s(x) = x + 1$ для кожного x;
- *функції вибору аргументів* (*селективні функції*): $I_m^{(n)}(x_1, x_2, ..., x_n) = x_m$ для всіх $x_1, x_2, ..., x_n$, $m = 1, 2, ..., n$; $n = 1, 2, ...$.

Функції вибору аргументів — це $I_1^{(2)}(x_1, x_2) = x_1$, $I_1^{(1)}(x_1) = x_1$, $I_2^{(2)}(x_1, x_2) = x_2$, $I_2^{(3)}(x_1, x_2, x_3) = x_2$ та інші.

Тепер уведемо оператори для отримання нових функцій із тих, що вже є.

Оператор суперпозиції. Нехай задано числові функції $f(x_1,..., x_m)$ і $g_1(x_1,..., x_n)$, $g_2(x_1,..., x_n)$, ..., $g_m(x_1,..., x_n)$, і нехай $h(x_1,..., x_n) = f(g_1(x_1,..., x_n),..., g_m(x_1,..., x_n))$. Тоді говорять, що функцію h одержано за допомогою *оператора суперпозиції* з функцій $f, g_1, g_2, ..., g_m$.

Приклад 9.5. Функцію $o(x_1, ..., x_n) = 0$ отримано за допомогою суперпозиції з функцій $o(x)$ та $I_1^{(n)}(x_1,..., x_n)$:

$$o(x_1, ..., x_n) = o(I_1^{(n)}(x_1, ..., x_n)).$$

Функцію $s_m^{(n)}(x_1, ..., x_n) = x_m + 1$ одержано за допомогою оператора суперпозиції з функцій $s(x)$ та $I_m^{(n)}(x_1, ..., x_n)$:

$$s_m^{(n)}(x_1, ..., x_n) = s((I_m^{(n)}(x_1, ..., x_n)).$$

Оператор примітивної рекурсії. Нехай $g(x_1, ..., x_{n-1})$ і $h(x_1, ..., x_{n-1}, x_n, x_{n+1})$ – дві числові функції, причому $n \geqslant 2$. Означимо третю функцію $f(x_1, ..., x_{n-1}, x_n)$ за допомогою такої схеми:

$$f(x_1, ..., x_{n-1}, 0) = g(x_1, ..., x_{n-1});$$
$$f(x_1, ..., x_{n-1}, y+1) = h(x_1, ..., x_{n-1}, y, f(x_1, ..., x_{n-1}, y)), y \geqslant 0.$$

Її називають *схемою примітивної рекурсії для функції* $f(x_1, ..., x_n)$ *за змінними* x_n та x_{n+1}; вона задає *примітивно рекурсивний опис* функції $f(x_1, ..., x_n)$ за допомогою функцій g і h. Говорять також, що функцію f отримано з функцій g й h за допомогою *оператора примітивної рекурсії* (за змінними x_n та x_{n+1}).

У разі примітивно рекурсивного опису функції $f(x)$, залежної від однієї змінної, схема примітивної рекурсії має такий вигляд:

$$\begin{cases} f(0) = a, \\ f(y+1) = h(y, f(y)), \quad y \geqslant 0, \end{cases}$$

де a – константа, $a \in N_0$. У такому разі говорять, що функцію $f(x)$ одержано за допомогою оператора примітивної рекурсії з константи a та функції h.

Приклад 9.6. Функцію $f_+(x, y) = x + y$ можна отримати за допомогою примітивної рекурсії з функцій $I_1^{(1)}(x)$ і $s_3^{(3)}(x, y, z)$. Справді,

$$f_+(x, 0) = x = I_1^{(1)}(x);$$
$$f_+(x, y+1) = f_+(x, y) + 1 = s_3^{(3)}(x, y, f_+(x, y)).$$

Функцію називають *примітивно рекурсивною*, якщо її можна одержати з найпростіших функцій за допомогою скінченної кількості застосувань операторів суперпозиції та примітивної рекурсії.

Отже, функції з прикладів 9.5 і 9.6 примітивно рекурсивні. Клас усіх примітивно рекурсивних функцій позначатимемо $K_{\text{пр}}$.

Продовжимо розглядати приклади примітивно рекурсивних функцій.

Приклад 9.7. Множення $f_\times(x, y) = xy$ примітивно рекурсивне:

$$f_\times(x, 0) = 0,$$
$$f_\times(x, y+1) = f_\times(x, y) + x = f_+(x, f_\times(x, y)).$$

Приклад 9.8. Піднесення до степеня $f_{\exp}(x, y) = x^y$ примітивно рекурсивне:

$$f_{\exp}(x, 0) = 1;$$
$$f_{\exp}(x, y+1) = x^y x = f_\times(x, f_{\exp}(x, y)).$$

Визначимо функцію $x \dot{-} y$ (арифметичне віднімання):

$$x \dot{-} y = \begin{cases} x - y, & \text{якщо } x > y; \\ 0 & \text{у протилежному випадку.} \end{cases}$$

Приклад 9.9. Функція $f(x) = x \dot{-} 1$ примітивно рекурсивна:

$$f(0) = 0; \quad f(y+1) = y.$$

Приклад 9.10. Функція $f(x, y) = x \dot{-} y$ примітивно рекурсивна:

$$f(x, 0) = x; \quad f(x, y+1) = x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 = f(x, y) \dot{-} 1$$

(тут використано функцію з прикладу 9.9).

Приклад 9.11. Функція

$$f(x, y) = |x - y| = (x \dot{-} y) + (y \dot{-} x)$$

примітивно рекурсивна.

Приклад 9.12. Функція

$$\mathrm{sg}(x) = \begin{cases} 0, \text{ якщо } x = 0, \\ 1, \text{ якщо } x \neq 0 \end{cases}$$

примітивно рекурсивна. Схема примітивної рекурсії тут така:

$$\mathrm{sg}(0) = 0;$$
$$\mathrm{sg}(x+1) = 1.$$

Приклад 9.13. Функції

$$\min(x, y) = x \dot{-} (x \dot{-} y),$$
$$\max(x, y) = y + (x \dot{-} y)$$

примітивно рекурсивні.

За допомогою функції sg із прикладу 9.12 та функції $\overline{sg} = 1 \dot{-} \mathrm{sg}$ побудуємо примітивно рекурсивний опис функцій, пов'язаних із діленням.

Приклад 9.14. Функція $r(x, y)$ — остача від ділення y на x — примітивно рекурсивна:

$$r(x, 0) = 0;$$
$$r(x, y+1) = (r(x, y) + 1)\mathrm{sg}(|x - (r(x, y) + 1)|).$$

Зміст другого рядка в схемі такий: якщо число $y + 1$ не ділиться на x, то

$$\mathrm{sg}(|x - (r(x, y) + 1)|) = 1, \quad r(x, y+1) = r(x, y) + 1;$$

якщо ж число $y + 1$ ділиться на x, то

$$r(x, y+1) = \mathrm{sg}(|x - (r(x, y) + 1)|) = 0.$$

Приклад 9.15. Функція $q(x, y) = \lfloor y/x \rfloor$ — частка від ділення числа y на x, тобто ціла частина дробу y/x — примітивно рекурсивна:

$$q(x, 0) = 0;$$
$$q(x, y+1) = q(x, y) + \overline{\mathrm{sg}}(|x - (r(x, y) + 1)|).$$

У другому рядку схеми другий доданок залежить від подільності числа $y + 1$ на x. Якщо воно ділиться на x, то $q(x, y + 1)$ на одиницю більше ніж $q(x, y)$; якщо ні, то $q(x, y + 1) = q(x, y)$.

Оператор мінімізації. Нехай $f(x_1, ..., x_{n-1}, x_n)$, $n \geqslant 1$ — числова функція. Означимо функцію $g(x_1, ..., x_{n-1}, x_n)$ так. Нехай $(m_1, ..., m_{n-1}, m_n)$ — довільний набір цілих невід'ємних чисел. Розглянемо рівняння

$$f(m_1, m_2, ..., m_{n-1}, y) = m_n. \tag{9.1}$$

Якщо воно має розв'язки в невід'ємних цілих числах, то візьмемо мінімальний із цих розв'язків і позначимо його μ_y. Якщо при цьому $f(m_1, ..., m_{n-1}, 0)$, ..., $f(m_1, ..., m_{n-1}, \mu_y - 1)$ визначено, то вважаємо, що

$$g(m_1, m_2, ..., m_{n-1}, m_n) = \mu_y.$$

Якщо ж або рівняння (9.1) не має розв'язків у невід'ємних цілих числах, або хоча б одне зі значень $f(m_1, ..., m_{n-1}, 0)$, ..., $f(m_1, ..., m_{n-1}, \mu_y - 1)$ невизначено, то $g(m_1, m_2, ..., m_{n-1}, m_n)$ невизначено.

Про функцію $g(x_1, ..., x_n)$, побудовану зазначеним способом, говорять, що її одержано з функції $f(x_1, ..., x_{n-1}, x_n)$ *застосуванням оператора мінімізації за змінною* x_n (або за допомогою *мінімізації за* x_n). У такому разі використовують позначення

$$g(x_1, ..., x_{n-1}, x_n) = \mu_y[f(x_1, ..., x_{n-1}, y) = x_n].$$

Оператор мінімізації називають також *μ-оператором.*

Функцію називають *частково рекурсивною*, якщо її можна отримати з найпростіших функцій за допомогою скінченної кількості застосувань операторів суперпозиції, примітивної рекурсії та мінімізації. Клас усіх частково рекурсивних функцій позначимо $K_{чр}$.

Частково рекурсивна функція може бути не всюди визначеною. Усюди визначену частково рекурсивну функцію називають *загальнорекурсивною.* Клас усіх загальнорекурсивних функцій позначатимемо як $K_{зр}$.

Кожна примітивно рекурсивна функція всюди визначена. Для множин $K_{пр}$, $K_{зр}$ та $K_{чр}$ виконуються співвідношення $K_{пр} \subset K_{зр} \subset K_{чр}$, причому всі включення строгі.

Приклад 9.16. Нехай $f(x) = x + 1$. За допомогою оператора мінімізації означимо функцію $g(x) = \mu_y[f(y) = x]$. Рівняння (9.1) набирає вигляду $y + 1 = x$. Отже, функцію $g(x)$ невизначено, якщо $x = 0$, і $g(x) = x - 1$, якщо $x > 0$.

Приклад 9.17. Доведемо, що функція

$$g(x_1, x_2) = \frac{x_1}{1 - x_1 x_2} \tag{9.2}$$

частково рекурсивна. Застосуємо оператор мінімізації за змінною x_1 до примітивно рекурсивної функції $f(x_1, x_2) = x_1(1 \dot{-} x_2)$. Тоді $g(x_1, x_2) = \mu_y[f(y, x_2) = x_1]$. Справді, рівняння (9.1) набирає вигляду

$$y(1 \dot{-} x_2) = x_1.$$

Отже,

$$g(x_1, x_2) = \begin{cases} 0, & \text{якщо } x_1 = 0, \\ x_1, & \text{якщо } x_2 = 0, \\ \text{невизначено, якщо } x_1 \neq 0 \text{ й } x_2 \neq 0. \end{cases}$$

або в аналітичному вигляді (9.2).

9.6. Теза Чорча. Зв'язок рекурсивних функцій із машинами Тьюрінга

Поняття частково рекурсивної функції — вичерпна формалізація поняття обчислюваної функції. Це можна подати як тезу А. Чорча (A. Church): *будь-яка функція, обчислювана якимось алгоритмом, частково рекурсивна.*

Теза Чорча — аналог тези Тьюрінга для рекурсивних функцій, але її було сформульовано раніше. Тезу Чорча, як і тезу Тьюрінга, не доводять. З їх зіставлення випливає твердження, яке являє собою теорему, тому її потрібно довести.

ТЕОРЕМА 9.3. Функція обчислювана за Тьюрінгом тоді й лише тоді, коли вона частково рекурсивна.

Доведення цієї теореми досить громіздке. Тут ми коротко опишемо лише його план.

Нехай функція частково рекурсивна. Потрібно довести, що вона обчислювана за Тьюрінгом. Спочатку доводять, що найпростіші функції обчислювані за Тьюрінгом (це досить очевидно), а потім — що оператори суперпозиції, примітивної рекурсії та мінімізації зберігають обчислюваність.

Нехай функція обчислювана за Тюрінгом. Потрібно довести, що вона частково рекурсивна. Таке твердження може видатися дуже слабким для того, щоб говорити про еквівалентність рекурсивних функцій, які мають справу із числами з розширеного натурального ряду, і машин Тьюрінга, які можуть не тільки обчислювати. Проте від будь-яких об'єктів можна перейти до числових за допомогою кодування, до того ж украй простого. З погляду машини, яка не вкладає в символи ніякого змісту, а лише виконує з ними задані операції. немає особливої різниці між числами та „нечислами". Свої елементарні дії — читання, запис, зсув, зміну стану — машина може виконувати за наявності на стрічці як цифр, так і інших символів. Різниця лише в інтерпретації отриманих результатів. Зокрема, ніщо не заважає інтерпретувати будь-який зовнішній алфавіт (алфавіт стрічки) $A = \{a_1, a_2, ..., a_m\}$ як множину цифр m-кової системи числення, а слово, записане на стрічці, — як m-кове число. Тоді будь-яку роботу машини Тьюрінга можна розглядати як перетворення чисел, тобто пошук значень числової функції. Саме таку інтерпретацію тут і беруть до уваги. Докладне доведення полягає в тому, щоб точно описати цю інтерпретацію (її називають *арифметизацією*) та показати, що будь-яке перетворення, реалізоване машиною Тьюрінга, частково рекурсивне, якщо інтерпретувати його як обчислення.

Контрольні запитання та завдання

1. Побудувати машину Тьюрінга, яка перетворює слово α на 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) $\alpha = 1^n 0 1^m$, $\beta = 1^m 0 1^n$.
2. Побудувати машину Тьюрінга, яка за будь-яким вхідним ланцюжком $0^n 1^m$ визначає, чи виконується умова $n = m$. Чи можна звідси дійти висновку, що машина Тьюрінга може більше, ніж скінченний автомат? Відповідь обґрунтувати.
3. Побудувати машину Тьюрінга, яка кожне слово $x_1 x_2 \ldots x_{n-1} x_n$ в алфавіті $\{a, b\}$ перетворює на слово $x_n x_{n-1} \ldots x_2 x_1$.
4. Побудувати машину Тьюрінга, яка обчислює *предикат симетрії* $S(x)$: для будь-якого слова $\alpha = x_1 x_2 \ldots x_{n-1} x_n$ в алфавіті $\{a, b\}$ $S(\alpha) = 1$, якщо $x_i = x_{n-i+1}$ для всіх $i = 1, 2, \ldots, n$, а ні, то $S(\alpha) = 0$.
5. Побудувати машину Тьюрінга, застосовну до слів 1^{3n} ($n \geqslant 1$) та незастосовну до слів 1^{3n+m}, де $n \geqslant 1$; $m = 1, 2$.
6. Побудувати машину Тьюрінга, яка обчислює числову функцію
   а) $I_3^{(4)}(x_1, x_2, x_3, x_4) = x_3$.
   б) $x \dot{-} y = \begin{cases} x - y, \text{ якщо } x > y, \\ 0 \quad \text{у протилежному випадку.} \end{cases}$
   в) $\operatorname{sg}(x) = \begin{cases} 0, & \text{якщо } x = 0, \\ 1, & \text{якщо } x \neq 0. \end{cases}$
   г) $\overline{\operatorname{sg}}(x) = \begin{cases} 1, & \text{якщо } x = 0, \\ 0, & \text{якщо } x \neq 0. \end{cases}$
   д) $f(x, y) = \min(x, y)$.
   е) $f(x) = 2x + 1$.
7. Побудувати машину Тьюрінга, яка обчислює числову функцію $f(x) = \lfloor x/2 \rfloor = m$, якщо $x = 2m$ або $x = 2m + 1$, $m \geqslant 0$.
8. Чи застосовні машини Тьюрінга T_1 і T_2, задані командами

$$T_1\colon q_1 1 \to 1 П q_1,\ q_1 0 \to 0 Н q_0,\ q_1 Л \to Л П q_1;$$
$$T_2\colon q_1 1 \to Л П q_1,\ q_1 0 \to 1 Л q_2,\ q_1 Л \to 1 Н q_0,$$
$$q_2 1 \to Л П q_1,\ q_2 0 \to 0 Л q_2,\ q_2 Л \to 0 П q_1,$$

до таких слів:
а) 1111111; б) 0110111; в) 111101; г) 001101?

9. Застосувавши оператор примітивної рекурсії за змінними x_2 та x_3 до функцій $g(x_1)$ і $h(x_1, x_2, x_3)$, одержати функцію $f(x_1, x_2)$. Записати функцію $f(x_1, x_2)$ аналітично:
   а) $g(x_1) = x_1$, $h(x_1, x_2, x_3) = x_1 + x_3$;
   б) $g(x_1) = x_1$, $h(x_1, x_2, x_3) = x_1 + x_2$;
   в) $g(x_1) = 2^{x_1}$, $h(x_1, x_2, x_3) = x_3^{x_1}$ (тут уважати, що $0^0 = 1$);
   г) $g(x_1) = 1$, $h(x_1, x_2, x_3) = x_3(1 + \operatorname{sg}|x_1 + 2 - 2x_3|)$;
   д) $g(x_1) = x_1$, $h(x_1, x_2, x_3) = (x_3 + 1)\operatorname{sg}\left(1 + \frac{x_3}{3}\right)$.
10. Застосувати оператор мінімізації до функції f за змінною x_i. Вислідну функцію подати аналітично:
   а) $f(x_1) = 3$, $i = 1$;
   б) $f(x_1) = \lfloor x_1/2 \rfloor$, $i = 1$;
   в) $f(x_1, x_2) = x_1 \div x_2$, $i = 1, 2$;
   г) $f(x_1, x_2) = I_1^{(2)}(x_1, x_2)$, $i = 2$;
   д) $f(x_1, x_2) = x_1 \div x_2$, $i = 1, 2$.
11. Застосувавши операцію мінімізації до відповідної примітивно рекурсивної функції, довести, що функція f частково рекурсивна:
   а) $f(x_1) = 2 - x_1$;
   б) $f(x_1) = x_1/2$;
   в) $f(x_1, x_2) = x_1 - 2x_2$.

Комп'ютерний проект

Скласти програму із зазначеними вхідними даними та результатами.

Дано машину Тьюрінга та слово в початковій конфігурації. Знайти слово в заключній конфігурації (уважати, що машина застосовна до слова, заданого в початковій конфігурації).