Обчислюваність: машини Тьюрінга та рекурсивні функції.

Власенко Дмитро Іванович
Курінний Григорій Чарльзович

27 грудня 2013 р.

Зміст

[1] Послідовне навчальне викладення обчислюваності можна знайти в книзі К.Катленд "Вычислимость. Введение в теорию рекурсивных функций." М.:Мир, 1983

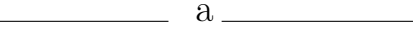
1 Машини Тьюрінга

1.1 Автомати

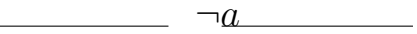
1.1.1 Автомати без пам’яті, релейно контактні схеми та схеми із функціональних елементів.

Автомат уявляємо як пристрій, який перетворює вхідні сигнали у вихідні. Найпростіший автомат однозначно реагує на вхідні символи, він не має пам’яті. Так вимикач кондиціонера реагує на сигнал від термометра, незалежно від того, якою була колись температура.

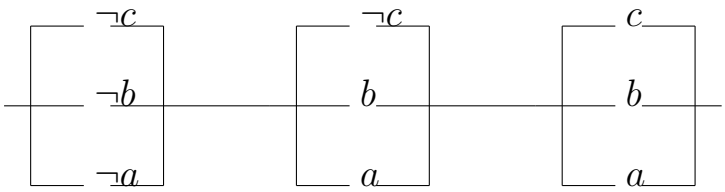
Якщо контакт замкнений при увімкненому реле a ми креслимо



а якщо контакт розімкнений при увімкненому реле, ми креслимо



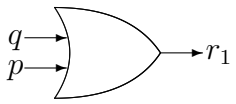
З'єднуючи контакти, що керуються реле, в схему, одержують релейно-контактну схему — це один із видів автоматів без пам'яті. Прикладом релейно контактної схеми буде

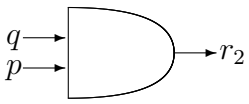


Шифрувальний пристрій може букви (елементи вхідного алфавіту) перетворювати на інші симоволи (елементи вихідного алфавіту), при цьому символ, на який змінюється буква, не залежить від попередньої роботи. Це також приклад автомата без пам'яті.

В релейно контактній схемі контакт, що керується реле, є неподільним елементм, матерівальне втілення якого дискретну математику не цікавить. Але є випадки, коли за неподільні елементи схем беруть більш складні прилади, їх називають функціональними елементами. Із функціональних елементів будують схеми із функціональних елементів. Функціональний елемент має так звані входи, на які подаються сигнали (елементи певної множини, вхідного алфавіту)

Функціоналний елемент має також вихід, на який він подає сигнал, елемент вихідного алфавіту. Найбільш важливими функціональними елементами є кон'юнктор, диз'юнктор та інвертор, в яких і вхідний і вихідний алфавіти складаються із 0 та 1. Схематичне зображення цих елементів дано на рис. 1,2, 3, а таблиці роботи вказані в табл. 1,2,





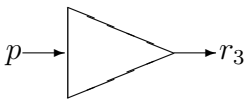


Рис. 1: Кон'юнктор Рис. 2: Диз'юнктор Рис. 3: Конвертор

3

p	q	r_1
0	0	0
0	1	0
1	0	0
1	1	1

Табл. 1: Таблиця роботи кон'юнктора

p	q	r_2
0	0	0
0	1	1
1	0	1
1	1	1

Табл. 2: Таблиця роботи диз'юнктора

p	r_3
0	1
1	0

Табл. 3: Таблиця роботи конвертора

1.1.2 Автомати із скінченною внутрішньою пам'яттю — із скінченною кількістю внутрішніх станів

Більш складний автомат має скінченну пам'ять, її називають внутрішніми станами. Так натискання кнопки на комп'ютері приводить до вмикання його, якщо він був вимкнений, і до вимикання, якщо був увімкнений.

Важливим для потреб шифрування є так званий регістр зсуву із лінійним зворотним зв'язком (РЗЛЗЗ). За допомогою такого регістру отримують послідовності 0 та 1, які важко відрізнити від повністю випадкових — такі послідовності називають псевдовипадковими.

РЗЛЗЗ уявляємо як пристрій, який

- має скінченну кількість (скажемо n) перенумерованих комірок пам'яті;
- Вміє прочитати вміст декількох комірок.

Перед початком роботи РЗЛЗЗ всі комірки тим чи іншим способом заповнюються нулями та одиницями. Регістр працює такт за тактом, і за кожним тактом створює вихідний символ — нуль або одиницю. Такт роботи полягає в наступному;

1. Створюється новий символ $a \in \{0, 1\}$ — цей символ дорівнює 1, якщо в тих комірках, які може читати РЗЛЗЗ, непарна кількість одиничок (сума за модулем 2 дорівннює 1), і 0 в протилежному випадку (сума за модулем 2 дорівнює 0).

2. Вміст першої комірки стає вихідним символом і перша комірка звільняється.

3. Вміст кожної комірки, крім першої, переноситься в комірку з на одиницю меншим номером.

4. в комірку з останнім номером записується створений символ a.

Розглянемо приклад. Нхай РЗЛЗЗ має 10 комірок пам'яті, в яких зберігаються булеві сталі 0 та 1, вміє читати вміст 3,4 і 7 комірки, Позначимо вміст відповідних комірок $x_1, \ldots, x_{10}$. Отже на кожному кроці створюється число $a = x_3 + x_4 + x_7$, створюється вихідний символ x_1 і відбувається зсув вмісту. Припустимо, що перед по чатком роботі вміст комірок такий

$$x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1, x_5 = 1, x_6 = 0, x_7 = 0, x_8 = 0, x_9 = 1, x_{10} = 1.$$

Тоді на першому кроці створюється вихідний символ $y_1 = x_1 = 0$; створюється символ $a = x_3 + x_4 + x_7 = 1 + 1 + 0 = 0$; вміст всіх комірок зсувається вліво в сусідню комірку (перша перед зсувом уже вивільнена)

$$x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 1, x_5 = 0, x_6 = 0, x_7 = 0, x_8 = 1, x_9 = 1;$$

в десяту комірку заноситься число a: $x_{10} = a = 0$. Цим перший крок завершився.

На другому кроці створюється вихідний символ $y_2 = x_1 = 0$; створюється символ $a = x_3 + x_4 + x_7 = 1 + 1 + 0 = 0$; вміст всіх комірок зсувається вліво в сусідню комірку (перша перед зсувом уже вивільнена)

$$x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 0, x_5 = 0, x_6 = 0, x_7 = 1, x_8 = 1, x_9 = 0;$$

в десяту комірку заноситься число a: $x_{10} = a = 0$. Цим другий крок завершився.

На третьому кроці створюється вихідний символ $y_3 = x_1 = 1$; створюється символ $a = x_3 + x_4 + x_7 = 1 + 0 + 1 = 0$; вміст всіх комірок зсувається вліво в сусідню комірку (перша перед зсувом уже вивільнена)

$$x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0, x_5 = 0, x_6 = 1, x_7 = 1, x_8 = 0, x_9 = 0;$$

в десяту комірку заноситься число a: $x_{10} = a = 0$. Цим третій крок завершився. Цей процес продовжується потрібну кількість разів. Після трьох кроків уже створена триелементна послідовність $y_1 = 1, y_2 = 1, y_3 = 1$.

1.1.3 Алан Тьюрінг, автомати із скінченною внутрішньою та потенційно нескінченною зовнішньою пам'яттю, алгоритми

Ще складніші автомати третього типу, в яких є скінченна кількість внутрішніх станів (внутрішня пам'ять) і потенційно нескінченна зовнішня пам'ять. Так комп'ютер має скінченну внутрішню пам'ять, і він може працювати з даними на дисках, флешках та подібному. В даному розділі нас будуть цікавити автомати третього типу. Математичних відповідників автоматів третього типу досить багато. Наразі зупиняємося на машинах Тьюрінга.

Тьюрінг Алан Матісон (23.06.1912 – 7.6.1954) в 1936-37 роках ввів формальне визначеня алгоритму та обчислюваної функції. Практично в той же час незалежне визначення алгоритма дав Еміль Леон Пост, використовуючи поняття абстрактної обчислювальної машини.

Пізніше визначеня алгоритма давали А.М.Колмогоров, А.А.Марков. Всі визначення виявилися еквівалентними, тобто все, що є агоритмом в одному визначенні є алгоритмом у всіх інших визначення. Еквівалентність незалежних визначень дозволяє стверджувати, що всі визначення дають "правильний", чи "адекватний" формальний відповідник інтуїтивному уявленню про алгоритм.

Перед тим, як щось говорити про формалізацію алгоритма (формалізація - відкидання несуттєвого) потрібно поговорити про наше інтуїтивне уявлення про нього.

Алгоритм уявляєтья як послідовність певних дій, які застосовуються до певних предметів. Окремі дії прості, їх набір обмежений, дії виконуються одна за одною, послідовно (по кроках). Предмети, до яких застосовується алгоритм, можна назвати іменами (перенумерувати, замінити, послідовносятями букв, знаків, цифр, ...) Те , що одержується в результаті, також можна назвати, позначити послідовністю символів. Сказане довзоляє розглядати лише алгоритми, що застосовуються до імен, тобто до послідовностей символів.

Множини, елементи яких використовують для утворення послідовностей — імен предметів, називають алфавітами. Елементи таких множин (алфавітів) називають буквами, а послідовності букв називають словами. Тепер ми можемо сказати. що алгоритми застосовують до слів із певного алфавітку. Цей алфавіт називають вхідним. Простіше всього іменувати предмети натуральними числами. Тому вхідний алфавіт звичайно пристосований до запису натуральних чисел.

Результат виконання алгоритму – також слово в якомусь алфавіті. Цей алфавіт називають вихідним. Вихідний алфавіт також звичайно пристосований до запису натуральних чисел.

Приписи, що, коли і як робити, виконання окремих простих дій (їх називають елементарними) записують ще в одному алфавіті — в алфавіті програми. Елементарних дій повинна бути скінченна множина.

Всі ці речі ми повинні побачити на прикладі формалізації алгори-

тма, який запропонував Алан Тьюрінг.

Зауважимо, що обходитися в агоритмах тільки додатніми натуральними числами незручно. Тому до множини натуральних чисел додається нуль. Отже далі вважаємо що числа беруться із розширеного натурального ряду $\overline{\mathbb{N}} = \mathbb{N} \cup \{0\}$.

1.2 Аксіоми, що стосуються алгоритмів

Аксіома запису Застосування алгоритму можна записати, і потім (за записом застосування алгоритму до того, до чого цей алгоритм можна застосувати) кожен однозначно може взнати — які були початкові дані, і який результат.

Аксіома програми Можна виділити множину програм (приписів), які всіма розуміються чітко, єдиним чином. При цьому все буде писатися в одному алфавіті. Кожен може відрізнити програму від непрограми.

Аксіома кодування Можна вважати, що алгоритми застосовуються до натуральних чисел і результатом їх застосування є натуральне число.

1.3 Загальні слова про обчислюваність, алгоритми та тези [2]

Мова йтиме про функції, чиї аргументи належать розширеному натуральному ряду — $\mathbb{N} \cup \{0\} = \overline{\mathbb{N}}$. Нас цікавитимусть функції, значення яких ми можемо обчислити в тому рзумінні, що є алгоритм, застосувавши який до аргументів ми через певну кількість кроків одержимо значення функції. Алгоритм розуміємо інтуїтивно. Такі функції називатимемо обчислюваними. Алгоритмів (вважаємо, що їх можна записати словами) зліченна кількість, а характеристичних функцій

[2] Послідовне навчальне викладення обчислюваності можна знайти в книзі К.Катленд "Вычислимость. Введение в теорию рекурсивных функций." М.:Мир, 1983

підмножин множини $\overline{\mathbb{N}}$ стільки ж, скільки підмножин у $\overline{\mathbb{N}}$ — більше ніж зліченна кількість. Тому є не обчислювані функції.

Формальні уточнення інтуїтивного поняття "обчислюваність" та "алгоритм" запропоновані в 1936 році (Гьодель, Ербран, Чьорч, Кліні, Тьюрінг, ...) Пізніше свої уточнення запропонували Пост, Марков, Шепердсон, Стерджіс,... Наш вибір того чи іншого підходу до обчислюваності суб'єктивний, — але це не важливо, оскільки є дослідження, які показують рівносильність усіх підходів. Всі підходи закінчуються тезою, яка стверджує,

> **Теза про обчислюваність** Будь-який інтуїтивний, неформальний алгоритм може бути модельований в запропонованому підході.

"Моделювання" означає заміну справжніх предметів, з якими має справу інтуїтивний алгоритм, на символи, з якими має справу підхід, а найпростіші дії інтуїтивного алгоритма над предметами, за міняються на найпростіші переходи від одних символів до інших. В такому мо вному оточенні часто вживається також слово "кодування". Кодування означає перезапис інформації з метою зручності обробітку, а в каналах зв'зку також для захисту від випадкового пошкодження. Тепер можна сказати уже досить складне речення — при моделюванні справжнього алгоритма початкові дані кодуються. Для прикладу, коли справжній алгоритм полягає в тому, щоб дві корови зігнати з трьома коровами в одну череду, то у відповідній машині Тьюрінга дві корови закодуються трьома одиницями, а три корови — чотирма одиницями. А резултьтатом виконання машини Тьюрінга, яка моделює наш алгорим, буде шусть одиниць на стрічці і машина Тьюрінга знаходиться у кінцевому стані.

Звичайно згадана теза носить ім'я того математика, який запропонував підхід. Таким чином, якщо підхід (моделювання машинами Тьюрінга) запропонував Тьюрінг, то згадана теза буде звучати так

Теза Тьюрінга Будь-який інтуїтивний, неформальний алгоритм може бути модельований на машині Тьюрінга,

Відповідно, є теза Чьорча, Поста, ...

Свідченнями на користь прийняття тези на віру, є такі

1. Різні незалежні дослідники запропонували незалежні підходи, які виявилися рівносильними — все, що можна зробити одним підходм, можна зробити іншими.

2. Для дуже багатьох інтуїтивно обчилюваних функцій доведена їх формальна обчислюваність, уже доведено, що дуже багато інтуїтивних алгоритмів м оделюються запропонованими підходами.

3. Ніхто не зумів знайти функцію, яка була б обчислювана в інтуїтивному розумінні, і яку не можна було б обчилити на машині Тьюрінга, чи за допомогою іншого (рівносильного) формального підходу, ніхто не зумів придумати інтуїтивно здійсненний алгоритм, який не можна було б здійснити на тій чи іншій моделі алгоритма.

Для математика з досвідом роботи з абгоритмами є додаткове свідоцтво на кор исть прийняття тези про обчислюваність, — це його власний досвід переведення інтуїтивного алгоритма у формальний. Для математика без досвіду роботи у царині алгоритмів прийняття чи не прийняття тези залежить від того, наскільки від може довіритися досвіду інших там, де його власний досвід відсутній чи просто малий.

1.4 Будова машини Тьюрінга — залізо і софт.

1.4.1 Будова машини Тьюрінга — залізо.

Машина Тьюрінга складається із трьох чистин - стрічки, голівки і системного блоку (керуючої частини).

Стрічка розділена на вічка, тобто можна вважати, що стрічка — це потенційно нескінченна вліво і вправо послідовність вічок (комірок пам'яті), в яких можна записати по одному символу із вхідного алфавіту. Вхідний алфавіт ми можемо позначити через

$$A = \{a_0, a_1, a_2, \ldots, a_n, \qquad n \geq 1.\}$$

В даному викладенні в прикладах будемо використовувати вхідний алфавіт із двох символів — 0 та 1. Вихідній алфавіт при використанні машин Тьюрінга збігається із вхідним.

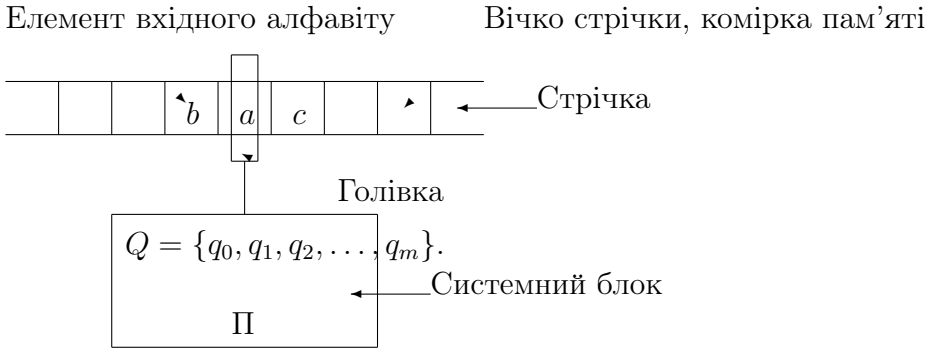


Рис. 4: Схематичне зображення машини Тьюрінга

Голівка напрямлена на вічко на стрічці. Голівка вміє прочитати вміст цього вічка, цієї ділянки стрічки, може при необхідності надрукувати в цьому вічку інший символ із вхідного а лфавіту, а також може рухатися вздовж стрічки. Елементарний (найпростіший) рух — на одне вічко вліво або на одне вічко вправо. Для позначння рухів використоуємо алфавіт рухів — множину із трьох букв

$$M = \{S, R, L\}$$

. Буква S символізує відсутність руху, буква L рух вліво на одне вічко, буква R — рух вправо на одне вічко.

Керуюча частина (системний блок) має скінченну скупність внутрішніх станів (вона утворює внутрішній алфавіт)

$$Q = \{q_0, q_1, q_2, \ldots, q_m, \} \quad m \geq 1,$$

і в ньому міститься програма П роботи машини див. рис. 4.

Системний блок містить програму. Програма — це набір команд, скінченна не порожня множина команд. Команда — це припис, який вказує, що робити машині у випадку, коли вона знаходиться у певному внутрішньому стані і голівка бачить певний символ на стрічці. А робити машина може наступне — може щось надрукувати в тому вічці, на яке дивиться, може зробити рух (або його не робити) і може перейти в інший внутрішній стан. Якщо команда записана у вигляді

$$aq \mapsto a'q'm, \qquad a, a' \in A, \quad q, q' \in Q, \quad m \in M,$$

то це означає, що у випадку, коли машина знаходиться у стані q, її голівка читає символ a, то голівка повинна в тому вічці, на яке дивиться, надрукувати символ a', перейти в стан q' і зробити рух m. Запис $1q_8 \mapsto 0q_3R$ і запис $q_81 \mapsto q_30R$ означають одне і те ж — якщо машина знаходиться у стані q_8 і голівка бачить 1, то голівка повинна замінити в тому вічку, на яке вона дивиться, 1 на 0, і просунутися на одне вічно направо, а системниий блок повинен перейти у стан q_3.

Поряд із повню словосполукою "машина Тьюрінга" будемо використовувати її абревіатуру МТ.

1.4.2 Запис даних і тлумачення результату роботи машини Тьюрінга — обчислення функції.

При створенні машин Тьюрінга, які обчислюють ту чи іншу функцію, потрібно домовитися про спосіб задання на стрічці аргументів функції, і про спосіб трактування результату роботи. Перезапис інформації з метою зручності подальшого обробітку називають кодуванням. Аргументами фунцій, які підлягають обчисленню, є натуральними числами. Перезапис цих даних за допомогою нулів та

одиниць є кодуванням. А сам запис набору змінних на стрічку за допомогою нулів та одиниць називають основним машинним кодом набору змінних. Якщо змінних n, і $\alpha = (x_1, x_2, \ldots, x_n)$ — набір змінних, $x_i \in \mathbb{N} \cup \{0\}, \quad 1 \leq i \leq n,$, то нехай $k(\alpha)$ — основний машинний код цього набору.

Машини Тьюрінга важливі з двох точок зору

- як абстрактну річ, яка вказує межі людський можливостей при обчисленнях;
- як практичну річ, на якій легко відпрацьовувати навички програмування.

З точки зору відпрацювання навичок програмування вхідний алфавіт повинен бути досить великий, закрема, він повинен включати цифри, символи незаповненості вічка, та подібне. З точки зору меж можливостей при обчисленнях, цікаво мати найпростіший вхідний алфавіт. Найпростіший алфавіт має лише два символи — 0 та 1. Ним і будемо користуватися.

1.4.3 Домовленості про основний машинний код набору змінних

При створенні основного машинного коду набору змінних користуються такими домовленостями.

Домовленість про межі запису Наявність двох суміжних комірок, що заповнені нулями, говорить про те, що основний машинний код набору змінних знаходиться повністю зліва від цих комірок, або повністю справа.

Домовленість про розділовий нуль Якщо комірка заповнена нулем, а обидві суміжні комірки заповнені одиницями, то нуль в цій комірці повідомляє про те, що він грає роль розділового знаку між окремими записами змінних в наборі.

Домовленість про запис нуля Оскільки запис нуля в комірці не несе повідомлення про ту чи іншу змінну, то нуль (аргумент функції) зписуємо як одну одиницю, що обмеженя зліва і справа нулями.

Домовленість про запис натурального числа Натуральне число $1 \leq n$ записуємо у вигляді $n+1$-єї одиниці, що йдуть одна за однією і обмежені зліва і справа нулями.

Домовленість про запис нульмісної функції Нульмісна функцыя це виділене число — натуральне число або нуль. Отже нульмісну функцію записуємо як відповідне число.

В наступній таблиці наведені приклади наборів змінних α та їх основних машинних кодів $k(\alpha)$.

α	$k(\alpha)$
$(2,3,5)$	$\ldots 0011101111011111100\ldots$
$(2,0,1,0)$	$\ldots 00111010110100\ldots$
0	$\ldots 00100\ldots$
$(0,1,3)$	$\ldots 0010110111100\ldots$

1.4.4 “Правильний“ початок і “правильний“ кінець роботи машини Тьюрінга.

Якщо немає точної вказівки про стан стрічки перед початком роботи, про стан, в якому машина починає роботу, про ту комірку, яку розглядає голівка на першому кроці своєї роботи. то маються на увазі наступні домовленості про “правильний“ початок і про “правильне “ завершення роботи створеної машини Тюрінга.

Домовленість про комірки за межами основного машинного коду До початку роботи стріка повінстю заповнена нулями — це якщо стрічка уявляється як справді (актуально) нескінченна. Якщо ж стрічка уявляється як потенційно (в можливості) нескінченна, то кожна нова комірка, яку розглядає голівка за межами основного машинного коду містить 0. Стрічка порожня, якщо всі її комірки заповнені нулями. Відповідно, непорожня ділянка стрічки та, в якій є одиниці. Вказівка на те,

1. як заповнена непорожня частина стрічки,

2. в якому стані знаходиться машина,

3. яку комірку розглядає голівка,

називається конфігурацією машини Тьюрінга.

Домовленість про початковий стан Початковий стан, якщо не сказане щось інше, позначається q_1. Перед першим кроком роботи машина знаходиться в початковому стані. Програма обов'язково містить команду, що примушує машину виконати певний крок в початковому стані. Перед по чатком роботи (на першому кроці своєї роботи) машина розглядає комірку, в якій міститься перша одиниця основного машинного коду набору змінних.

Приклад запису вектора $\alpha = (x_1, x_2, x_3)$ з "правильним" розташуванням голівки:

$$000\overset{\downarrow}{\underbrace{1\,11\ldots1}_{x_1+1}}0\underbrace{11\ldots1}_{x_2+1}0\underbrace{11\ldots1}_{x_3+1}00 \tag{1}$$

Приклад запису $(2, 1, 0, 3)$, причому голівка читає третю одиницю в x_1:

$$0011\overset{\downarrow}{1}011010111100, \tag{2}$$

таке розташування голівка не "правильне".

Запис того, що машина розглядає певну одиничку на стрічці за допомогою стрілочки є "двоповерховим", малозручним. Тому часто використовується наступна домовленість: записуємо назву стану, в якому знаходиться машина, перед коміркою, яку розглядає голівка (зліва від цієї комірки). Отже машина правильно починає роботу з обчислення функції від (3,0,0,2), якщо маємо конфігурацію

$$00q_1 11110101011100.$$

В записі конфігурацій використовуються повторювачі — верхні індекси, що показують кількість повторень симовла чи групи символів. Якщо відомо, що МТ знаходиться в стані q, то конфігурацію (1) може бути записана у вигляді

$$00q1^{x_1+1}01^{x_2+1}01^{x_3+1}00,$$

конфігурацію (2) може бути записана у вигляді

$$001^2q101^20101^400,$$

а конфігурацію $00q01110111011100$ можна записати у вигляді $00q(01^3)^300$.

Приклад конфігурації: $0011q_710100$ — на стрічці знаходиться слово 11101, машина знаходиться в стані q_7, голівка читає третю одиницю (що машина читає — записано справа від запису стану). Ще один запис $01^xq_101^y$ — на стрічці записано x та y одиничок, що роз'єднані знаком 0. Голівка знаходиться над нулем, який роз'єднує групи одиниць, машина знаходиться в стані q_1.

Домовленість про кінцевий стан

В сукупності внутрішніх станів виділяється один або декілька кінцевих станів. Кінцевий стан відрізняється від інших тим, що команди, які б вказували, що робити в кінцевому стані, відсутні. Звичайно, для позначення одного кінцевого стану використовується q_0. Якщо кінцевих станів деквлька, то використовують позначення $q_{01}, q_{02}, q_{03}, \ldots, q_{0n}$.

Нехай $y = f(x_1, \ldots x_n)$ — деяка функція і $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ — набір змінних, МТ обчислює часткову функцію f якщо

у випадку, коли α належить області визначення, тоді

1. машина закінчує роботу в кінцевому стані;
2. на стрічці знаходиться основний код числа y;
3. голівка розташована над першою одиничкою коду,

а у випадку, коли α не належить області визначення функції f, тоді при роботі машини створиться одна із ситуацій

1. створюється конфігурація, коли машина не знаходиться в кінцевому стані, але команда, відповідна даній конфігурації, відсутня (кажуть — машина зависла);
2. машина нескінченно продовжує роботу;

3. машина закінчує роботу в кінцевому стані, але на стрічці лишається не основний машинний код числа.

Ситуацію, коли машина закінчує роботу в кінцевому стані, на стрічці основний код набору змінних, але голівка не над першою одиницею, не розглядаємо, оскільки таку машину можна видозмінити, і голівка буде над першою одиничкою.

1.5 Програми. Обчислюваність функцій і предикатів.

1.5.1 Програми. Обчислюваність конкретних функцій.

Перший приклад програми, це програма, яка "правильно"розташовує голівку — машина повинна почати роботу на стрічці, де записаний машинний код (набір змінних), і не змінюючи на стрічці нічого закінчує роботу в кінцевому стані над першою одиничкою коду.

Приклад 1.1 *Створимо програму знаходження лівої одиниці основного коду у випадку, коли на стрічці знаходиться основний код, а голівка знаходиться будь-де на стрічці.*

Вважаємо, для зручності, що рухається голівка, а стрічка лишається на місці. У початковому стані q_1 машина перевіряє, який символ розглядає голівка

$$q_1 1 \to q_2 1 L, \quad q_1 0 \to q_3 0 L.$$

У стані q_2 машина знає, що на стрічі немає нічого зайвого (крім основного машинного коду), голівка читає символ всередині коду (розташована над кодом), машині потрібно шукати першу одиницю коду зліва.

У стані q_3 машина ще не знає, де знаходиться код. 0, який вона розглядає, може бути і розділовим нулем, і нулем поза кодом як зліва так і прсправа. Машина пробує знайти код на сусідній зліва ділянці.

У стані q_3 машина розглядає символ зліва від початкового і усуває певну невизначеність, — якщо бачить одиницю, то голівка уже розташована над кодом, а якщо нуль, то голівка розташована поза кодом, і машина починає створювати мітки для переглянутої частини стрічки.

$$q_3 1 \to q_2 1 L, \quad q_3 0 \to q_4 1 R.$$

q_4 означає, що голівка знаходиться поза межами коду і починає створення двох міток (двох одиниць, якими вона буде відмічати ліву та праву границю переглянутої частини стрічки. Ліва мітка-одниця уже створена.

$$q_4 0 \to q_5 1 L.$$

У стані q_5 машина знає, що уже створені дві одиниці-мітки для переглянутої частини стрічки, і машина бажає рухатися наліво для відшукування лівої одиниці-мітки і перегляду можливості пересунути ліву мітку на крок вліво.

$$q_5 1 \to q_6 0 L, \quad q_5 0 \to q_5 0 L.$$

q_6 означає, що ліва мітка знайдена, знищена і голівка зсунута ще на один символ наліво для дослідження можливості пересунути мітку.

$$q_6 1 \to q_{13} 1 R, \quad q_6 0 \to q_7 1 R.$$

q_{13} означає, що зліва від початкового стану знайдено код, але шукати першу його одницю рано — потрібно спочатку знищити створену праву мітку.

q_7 означає, що код ще не знайдено, ліва мітка пересунута, машина повинна йти до правого краю переглянутої частини стрічки і пошукати код справа.

$$q_7 1 \to q_8 0 R, \quad q_7 0 \to q_7 0 R.$$

У стані q_8 машина знайшла праву мітку, знищила її і пробує пересунути її направо, досліджуючи символ справа від розглянутої частини ділянки.

$$q_81 \to q_91L, \quad q_80 \to q_51L.$$

q_9 означає, що знайдено код (першу його одиницю) справа від початкового стану, але зупинятися не можна, оскільки на стрічці лишилася зайва ліва мітка-одиниця.

$$q_91 \to q_{10}0R, \quad q_90 \to q_90L.$$

У стані q_{10} машина знищила всі мітки, знає, що код знаходиться справа, голівка повинна рухатися направо до першої одинички.

$$q_{10}1 \to q_01S, \quad q_{10}0 \to q_{10}0R.$$

Далі розберемося, що повинна робити машина у стані q_{13}, коли вона знайшла код зліва, але лишилася права мітка.

$$q_{13}1 \to q_{14}0L, \quad q_{13}0 \to q_{13}0R.$$

У стані q_{14} машина знає, що мітки знищені, код знаходиться зліва від початкового стану. Машина повинна знайти цей код. Коли машина його знайде, вона повинна перейти у стан q_2

$$q_{14}1 \to q_21L, \quad q_{14}0 \to q_{14}0L.$$

Тепер потрібно розібратися ір випадком, коли машина знаходиться над символом коду, міток ніяких немає, і їй потрібно знайти ліву одиницю.

$$q_21 \to q_21L, \quad q_20 \to q_{11}0L.$$

У стані q_{11} машина шукає кінець коду, знайшла 0, але сумнівається, — це кінець коду чи просто розділовий нуль.

$$q_{11}1 \to q_2 1L, \quad q_{11}0 \to q_{12}0R.$$

Стан q_{12} означає, що машина проскочила першу одиницю, їй потрібно повернутися назад і закінчити роботу:

$$q_{12}0 \to q_0 0R.$$

Об'єднавши виписані команди в одне ціле одержуємо програму. Порядок коман в програмі значення не має. Отже програмою машини Тюрінга, яка шукає ліву одиницю, буде

$$q_1 1 \to q_2 1L, \quad q_1 0 \to q_3 0L, q_3 1 \to q_2 1L, \quad q_3 0 \to q_4 1R, q_4 0 \to q_5 1L, q_5 1 \to q_6 0L, \quad q_5 0 \to$$

$$q_6 1 \to q_{13} 1R, \quad q_6 0 \to q_7 1R, q_7 1 \to q_8 0R, \quad q_7 0 \to q_7 0R, q_8 1 \to q_9 1L, \quad q_8 0 \to q_5 1L, q_9 1 \to$$

$$q_9 0 \to q_9 0L, q_{10} 1 \to q_0 1S, \quad q_{10} 0 \to q_{10} 0R, q_{13} 1 \to q_{14} 0L, \quad q_{13} 0 \to q_{13} 0R, q_{14} 1 \to q_2 1L,$$

$$q_{14} 0 \to q_{14} 0L, q_2 1 \to q_2 1L, \quad q_2 0 \to q_{11} 0L, q_{11} 1 \to q_2 1L, \quad q_{11} 0 \to q_{12} 0R, q_{12} 0 \to q_0 0R.$$

В такому масивы команд потрыбну команду шукати важко. Зручніше всі команди записуати у вигляді таблиці — на перетині рядка, що помічений симвоом вхідного алфавіту, та стовпчика, що помічений певним стаом, виписується припис — що повинна робити машина, яка має вказаний стан і бачить вказаний символ. Запишемо одержану програму знаходження першої одиниці основного машинного коду у вигляді таблиці.

	1	2	3	4	5	6	7
	q_1	q_2	q_3	q_4	q_5	q_6	q_7
0	$q_3 0R$	$q_{11} 0L$	$q_4 1R$	$q_5 1L$	$q_5 0L$	$q_7 1R$	$q_7 0R$
1	$q_2 1L$	$q_2 1L$	$q_2 1L$	$q_2 1L$	$q_6 0L$	$q_{13} 1R$	$q_8 0R$

(3)

	8	9	10	11	12	13	14
	q_8	q_9	q_{10}	q_{11}	q_{12}	q_{13}	q_{14}
0	$q_5 1L$	$9_9 0L$	$q_{10} 0R$	$q_{12} 0R$	$q_0 0R$	$q_{13} 0R$	$q_{14} 0L$
1	$q_9 1L$	$q_{10} 0R$	$q_0 1S$	$q_2 1L$		$q_{14} 0L$	$q_2 1L$

(4)

Приклад 1.2 *Припустимо, що на стрічці знаходиться основний машинний код набору (1,2) і машина знаходиться в конфігурації*

$$0011011100q_100,$$

тобто машина знаходиться в початковому стані і голівка напрямлена на третій 0 після останньої одиниці. Потрібно застосувати машину Тьюрінга з з програмою (3),(4) до цієї конфігурації.

виписуємо послідовно ті конфігурації, які виникають після кожного виконання належної команди.

1. $0011011100q_100$
2. $00110111000q_300$
3. $001101110001q_400$
4. $00110111000q_51100$
5. $0011011100q_600100$
6. $00110111001q_70100$
7. $001101110010q_7100$
8. $0011011100100q_800$
9. $001101110010q_5010$
10. $00110111001q_50010$
11. $0011011100q_51010$
12 $001101110q_600010$
13. $0011011101q_70010$
14 $00110111010q_7010$
15 $001101110100q_710$
16 $0011011101000q_80$
17. $001101110100q_50100$
18. $00110111010q_500100$
19 $0011011101q_5000100$
20 $001101110q_51000100$
21. $00110111q_600000100$
22. $001101111q_70000100$
23. $0011011110q_7000100$
24 $00110111100q_700100$
25 $001101111000q_70100$
26 $0011011110000q_7100$
27 $00110111100000q_800$
28 $0011011110000q_5010$
29 $001101111000q_50010$
30 $00110111100q_500010$
31 $0011011110q_5000010$
32 $001101111q_50000010$
33 $00110111q_510000010$
34 $0011011q_6100000010$
35 $00110111q_{13}00000010$
36 $001101110q_{13}0000010$
37 $0011011100q_{13}000010$
38 $00110111000q_{13}00010$
39 $001101110000q_{13}0010$
40 $0011011100000q_{13}010$
41. $00110111000000q_{13}10$
42. $0011011100000q_{14}0$
43. $001101110000q_{14}00$
44. $00110111000q_{14}00$
45. $0011011100q_{14}00$
46. $001101110q_{14}00$
47. $00110111q_{14}00$
48. $0011011q_{14}100$
49. $001101q_21100$
50. $00110q_211100$
51. $0011q_2011100$
52. $001q_{11}1011100$
53. $00q_211011100$
54. $0q_2011011100$
55. $00q_{11}0011011100$
56. $0q_{12}011011100$
57. $0q_011011100$

Наявність згаданої програми дозволяє не перейматися місцем, де машина закінчила роботу (позиція на стрічці, яка розглядається го-

лівкою), тому що невелика видозміна програми за допомогою згаданої підпрограми дасть нам машину Тьюрінга, яка закінчує роботу в належній позиції (розглядаючи першу одиничку).

У випадку, коли зустрілася команда $q_2 1 \to q_2 1 S$, вважаємо, що машину "зациклило", вона не закінчила роботу, — машина повинна закінчити роботу виключно в кінцевому стані, і не повинно існувати команди, яка б стосувалася машини у кінцевому стані (хоч фактично машина зупинилася).

Будемо вважати, що початковий стан у машини один (зазвичай це q_1, але при необхідності ми можемо його перепозначати) а кінцевих один — q_0 або декілька — $q_{01}, q_{02}, \ldots\ q_0', q_0'', \ldots$. Наявність кількох кінцевих станів важлива при обчисленні предикатів та при передачі результатів роботи однієї машини іншим машинам.

Приклад 1.3 *Створимо MT, яка обчислює функцію* $x + y$.

Область визначення функції, яку будуємо, є множина всіх пар (x, y) чисел із розширеного натуральнго ряду. Тому передбачається, що на стрічці є послыдовність із $x + 1$ одиниці і послідовність із $y + 1$ одиниці, що розділені одиним нулем. Голівка розташована над першою із записаних одиниць і машина знаходиться в початковому стані q_1.

Машина повинна закінчити роботу в кінчевому стані q_0, на стрічці в цей час повинна бути послідовність із $x + y + 1$-єї одиниці, а голівка повинна розталовуватися над першою із одиниць.

Потрібною програмою буде

	q_1	q_2	q_3
0	$q_2 1 L$	$q_3 0 R$	
1	$q_1 1 R$	$q_2 1 L$	$q_0 0 R$

Функція з аргументами із розширеного натурального ряда називається обчислюваною (на машині Тьюрінга) якщо існує машина Тьюрінга, яка обчислює цю функцію.

Теорема 1.1 (Теорема про обчислюваність найпростійших функцій) *Функції*

$$0(x_1, x_2, \ldots, x_n) = 0, \quad s(x) = x + 1, \quad I_m^n(x_1, x_2, \ldots, x_n) = x_m$$

обчислювані на машині Тьюрінга.

Доведення. Доведення теореми конструктивне — створюються програми.

■

1.5.2 Предикати, їх обчислюваність

> $n-$місним (або $n-$арним)предикатом на множині називають підмножину $n-$го декартового степеня ($n \geq 1$) цієї множини. Символ, яким позначається предикат, називається предикатним символом. 0-місні предикати це висловлювання, які не мають зміних.

Неформально предикатом називають висловлювання із змінними (стіл x дерев'яний, сма двох чисел парна, студент x сидить між студентами y та z. Якщо замість змінних підставити предмети, то висловлення стане або істинним, або хибним. Якщо змінних n то предикат $n-$місний. Множина, на якій задано предикат (столи, числа, студенти) визначається мовним оточенням. Послідовності значень змінних, на яких предикат істинний, утворюють множину, яку називають областю істиності предиката. Символ, яким позначають предикат, також називають предикатом. Вся ця неформальна нечіткість не призводить до помилок (якщо не захотіти їх створити навмисно.) Якщо ж непорозуміння все ж виникли із визначенням, тоді потрібно використовувати формальне визначення.

> Предикат $P(x_1, \ldots x_n)$ обчислюваний (на машині Тьюрінга), якщо існує машина Тьюрінга, яка починаючи роботу із основного машинного коду набору змінних закінчує роботу

в одному із двох кінцевих станів у залежності від істиності предиката на цьому наборі.

Приклад 1.4 *Приклад обчислюваного предиката — предикат "число x парне".*

Щоб довести обчислюваність, потрібно створити машину Тьюрінга (написати програму), яка б обчислювала цей предикат. Створимо програму.

Отже наша машина розрахована на те, що на стрічці знаходиться основний машинний код числа x. Передбачаємо початковий стан q_1 і два кінцевих стани — q_0', q_0''. Будуємо машину так, щоб вона закінчила роботу у стані q_0', коли число парне, і в стані q_0'', коли число непарне.

Стан q_2 буде нам сигналізувати, що машина уже бачила непарну кількість одиниць, а стан q_1 буде нам сигналізувати, що машина уже бачила парну кількість одиниць. Отже шуканою програмою буде

	q_1	q_2	q_3	q_4
0	q_3L	q_40L	$q_0'0R$	$q_0''0R$
1	q_21R	q_11R	q_31L	q_41L

З кожним предикатом $P(x_1, x_2, \ldots x_n)$ пов'язується його характеристична функція $f_P(x_1, x_2, \ldots, x_n)$.

Характеристична функція дорівнює одиниці на тих наборах змінних, на яких предикат істинний, і дорівнює нулю на тих наборах, на яких цей предикат хибний.

Оскільки машин Тьюрінга зліченна множина, а підмножин множини натуральних чисел незліченна множина, то існують необчислювані предикати.

Теорема 1.2 (про обчислюваність найпростіших предикатів) *Предикати $x = y$, $x < y$, $x = 0$ обчислювані.*

Доведення. Доведення конструктивне. Програмою обчислення предиката $x = 0$ буде

	q_1	q_2
0		$q_{01}L$
1	$q_2 1R$	$q_{02}L$

Кінцевий стан q_{01} відповідає тому, що предикат "$x = 0$" істинний, а кінцеий стан q_{02} відповідає тому, що предикат "$x = 0$" хибний.

■

1.6 Організація роботи декількох машин на одній стрічці.

Одна із можливостей машин Тьюрінга — об'єднання в одній машині (на одному залізі) кількох — застосування софту кількох машин. Таким чином є одна стрічка, одна голівка, один системний блок, а потрібно організувати на них роботу кількох машин, що мають свої внутрішні стани, свої програми.

1.6.1 Композиція машин Тьюрінга

Перший випадок випадок організації роботи двох машин, коли з даними (з усіма), що на стрічці, працює одна машина, і коли вона закінчує роботу, з усіма новими даними працює інша машина. Цей випадок називаємо композицією двох машин.

Беремо дві машини T_1, T_2 з програмами P_1, P_2. Вважаємо, що їх внутрішні алфавіти Q_1, Q_2 не перетинаються. Нехай $q' \in Q_1, q'' \in Q_2$ відповідно якийсь кінцевий стан машини T_1 і початковий стан машини T_2. Будуємо нову машину $T = T_1T_2$. Її внутрішній алфавіт $Q_1 \cup Q_2 \setminus \{q'\}$. Для побудови програми P нової машини (яку називають композицією машин Тьюрінга T_1, T_2) об'єднують програми P_1, P_2 і скрізь заміняють q' на q''. Серед внутрішніх станів нової машини T немає початкового стану машини T_2 — він замінений на кінчевий стан машини T_1. Початковим станом побудованої машини T є початковий

стан машини T_1, з нього й починається робота нової машини. Кінцевими станами нової машини є всі кінчеві стани машини T_2 і ті кінцеві стани машини T_1, що відрізняються від q'.

Якщо не очевидно, які стани ототожнюються, і, відповідно, яка машина починає діяти першою, то це по трібно вказувати окремо.

Приклад 1.5 *Машини T_1 має початковий стано q_1 і кінцеві стани q_{01}, q_{02}, машини T_2 має початковий стан q_3 та кінцеві стани q_{03}, q_{04}.*

Ототожнюємо початковий стан q_3 з кінцевим станом q_{02}, одержуємо композицію T_1T_2

T_1

	q_1	q_2
0	q_{01}	q_1
1	q_{02}	q_2

T_2

	q_3	q_4
0	q_4	q_{03}
1	q_{04}	q_4

T_1T_2

	q_1	q_2	q_3	q_4
0	q_{01}	q_1	q_4	q_{03}
1	q_3	q_2	q_{04}	q_2

Зауважимо, коли одна машина має n_1 станів k_1 команд, а друга машина має n_2 станів k_2 команд, то композиція цих машин має $n_1 + n_2 - 1$ станів $k_1 + k_2$ команд.

1.6.2 Ітерація машин Тьюрінга

> Ітерація машини Тьорінга полягає в тому, що один із її кінцевих станів ототожнюється з некінцевим (початковим, або просто зручним) станом.

Зауважимо, коли одна машина має n_1 станів k_1 команд,то ітерація цієї машини має $n_1 - 1$ станів k_1 команд. Отже композиція машини самої з собою буде виконуати ту ж роботу, буде обчислювати те ж саме, що і ітерація, але композиція буде мати вдвічі більше команд.

За допомогоо ітерації організовуються циклічні обчислення, цикли.

Приклад 1.6 *Побудувати ітерацію машини Тьюрінга*

1.6.3 Організація роботи машини на гратці

Ми уже знаємо, як записати на стрічці дані (основний ашинний код даних), тепер покажемо, як можна записати дані для декількох машин так, щоб ці дані можна було легко розрізняти. Це робиться за допомогою так званого граткового коду, квазіосновного коду.

Спочатку на стрічці виділяємо одне вічко і кажемо, що це вічко має номер 0. Після цього всі вічка одержують номери — цілі числа. Гратка із кроком l — послідовність ділянок (вічок) стрічки, номери яких порівнювані за модулем l, тобто різниця номерів ділиться на l. Для прикладу, якщо на стрічці маємо 16 ділянок із записами

0	1	0	0	1	0	0	1	0	0	0	0	0	1	0	0

то ділянки з номерами 2,5,8,11,14 знаходяться на гратці з кроком 3, на цій гратці стоїть основний машинний код

1	1	1	0	1

набору (2,0).

Якщо основний машинний код набору чисел записати (послідовно) на гратці, то те, що одержиться, називають квазіосновним кодом.

Приклад 1.7 *На стрічці є основний машинний код набору (2,3), тобто 0011101111000. Потрібно створити машину, яка б переписала цей основий код в квазіосновний код на гратку з кроком 2, залишивши другу гратку порожньою, тобто ця машина повинна записати на стрічці*

00001010100010101010000

Приклад 1.8 *На стрічці є основний машинний код набору (2,3), тобто 0011101111000. Потрібно створити машину, яка б продублювала цей основий код в два квазіосновних коди на дві гратки з кроком 2, тобто ця машина повинна записати на стрічці*

00001111110011111111110000

Процес перезапису основного коду на гратку, тобто перехід до квазіосновного коду, оборотний — від квізіосновного коду на гратці з кроком $n, n > 1$ можна перейти до основного коду набору, знищивши всі записи на решті граток з цим кроком.

Приклад 1.9 *На стрічці є гратка з кроком 2 і на цій гратці записаний квазіосновний код набору (2,3), друга гратка порожня, тобто на стрічці є запис 000010101000010101010000. Потрібно створити машину, яка б створила на стрічці основний код набору (2,3), тобто ця машина повинна записати на стрічці*

001110111100

Теорема 1.3 (про організацію роботи машини на гратці) *: Для будь-якої машини M можна побудувати нову машину M' таку, що*

1. *За межами гратки M' не змінює записи;*

2. *На гратці M' виконує ту ж роботу, що і задана машина M.*

Доведення. Дано: Машина M — тобто зовнішній алфавіт $A = \{0, 1\}$, внутрішній алфавіт Q і рухи $\{R, L, S\}$. Машина M має програму. Є гратка з кроком n.

Будуємо нову машину з тим же зовнішнім алфавітом $\{0, 1\}$. Для кожного внутрішнього стану q створюємо $2 \cdot (n-1)$ нових станів $r_{iq}, i = 1, 2, \ldots n-1$ та $l_{iq}, i = 1, 2, \ldots n-1$. Уявляємо, що у стані r_{iq} та l_{iq} машина пам'ятає кількість i та стан q.

Для нової програми беремо всі команди старої програми, які мають вигляд $qa \to q'a'S, \quad q, q' \in Q, \quad a, a' \in \{0, 1\},$.

За кожною командою вигляду $qa \to q'a'L, \quad q, q' \in Q, \quad a, a' \in \{0, 1\}$ створюємо n нових команд

$$qa \to l_{1q'}a'L, \quad l_{i-1,q'}x \to l_{iq'}xL, \quad 2 < i < n, \qquad l_{n-1,q'}x \to q'xL,$$

За кожною командою вигляду $qa \to q'a'R, \quad q, q' \in Q, \quad a, a' \in \{0, 1\}$ створюємо n команд

$$qa \to r_{1q'}a'R, \quad r_{i-1,q'}x \to r_{iq'}xR, \quad 2 < i < n, \qquad r_{n-1,q'}x \to q'xR,$$

Створені нові команди утворюють нову потрібну програму.
Доведення закінчене.

■

1.6.4 Розгалуження машин Тьюрінга

Приклад 1.10 *Побудувати МТ, яка обчислює функцію*

$$f(x) = \begin{cases} 0, & \textit{якщо } x \textit{ дорівнює } 0; \\ x-1, & \textit{якщо } x \textit{ додатнє непарне число}; \\ x-2, & \textit{якщо } x \textit{ додатнє парне число.} \end{cases} \tag{5}$$

Створюємо чотири машини Тьюрінга M_i $i = 1, 2, 3, 4$:

M_1 обчислює предикат "$x = 0$";

M_2 обчислює предикат "x є непарним числом"

M_3 обчислює функцію $x - 1$;

M_4 обчислює функцію $x - 2$.

M_1

	q_1	q_2
0		$q_{01}L$
1	$q_2 1R$	$q_{02}L$

, M_2

	q_1	q_2	q_3	q_4
0	$q_3 L$	$q_4 0L$	$q_{01}0R$	$q_{02}0R$
1	$q_2 1R$	$q_1 1R$	$q_3 1L$	$q_4 1L$

,

прогама машини M_3 складається із однієї команди $q_1 1 \to q_0 0S$, а програма машини M_4 має дві команди $q_1 1 \to q_2 0R$, $\quad q_2 1 \to q_0 0S$,

Робимо так, щоб різні машини мали різні стани — до індексів стані додаємо індекс машини. Таким чином, внутрішніми станами машини M_1 тепер будуть $q_{11}, q_{21}, q_{011}, q_{021}$, внутрішніми станами машини M_2 тепер будуть $q_{12}, q_{22}, q_{32}, q_{42} q_{012}, q_{0221}$, внутрішніми станами машини M_3 тепер будуть q_{13}, q_{23}, q_{03}.

Створюємо композицію $M_1 M_2$ машин M_1, M_2 за станами q_{01}, q_{12}, тобто ототожнюємо ці стани:

$$q_{021} = q_{12} = q'.$$

Таким чином одержуємо машину $N = M_1 M_2$ з програмою

	q_{11}	q_{21}	q'	q_{22}	q_{32}	q_{42}
0		$q_{011}0L$	$q_{32}L$	$q_{42}0L$	$q_{012}0R$	$q_{022}0R$
1	$q_{21}1R$	$q'1L$	$q_{22}1R$	$q'1R$	$q_{32}1L$	$q_{42}1L$

,

Створена машина N має три кінцеві стани q_{01}, q_{012}, q_{022}:

стан q_{01} сигналізує про те, що число x дорівнює нулю;

стан q_{012} сигналізує про те, що число x не дорівнює нулю і парне;

стан q_{022} сигналізує про те, що число x не дорівнює нулю і непарне.

На останньому кроці створюємо одночасно дві композиції машини N — з машиною M_3 за станами q_{022} і q_{13}, та з машиною M_4 за станами q_{012} і q_{14}, тобто ототожнюємо стани:

$$q_{022} = q_{13} = q'', \qquad q_{012} = q_{14} = q'''',$$

і об'єднуємо програми

	q_{11}	q_{21}	q'	q_{22}	q_{32}	q_{42}	q''	q''''	q_{24}
0		$q_{011}0L$	$q_{32}L$	$q_{42}0L$	$q''''0R$	$q''0R$			
1	$q_{21}1R$	$q'1L$	$q_{22}1R$	$q'1R$	$q_{32}1L$	$q_{42}1L$	q_{03}	q_{24}	q_{04}

Одержана програма і є потрібною.

Одночасна композиція машини з двома або більшою кількістю МТ називається розгалуженням цієї машини. При розгалуженні потрібно вказувати, за якими станами відбувається композиція.

Теорема 1.4 *Предикат обчислюваний на машині Тьюрінга тоді і тільки тоді, коли обчислювана його характеристична функція.*

Доведення.

Припустимо, що предикат $P(x_1, x_2, \ldots, x_n,)$, $n \geq 2$. Відомо, що його характеристична функція $f(x_1, x_2, \ldots, x_n)$ обчислювана. Доведемо обчислюваність предиката P.

Обчслюваність характиристичної функції f означає, що є машина Тьюрінга M, яка починаючи роботу зі стрічки з записом основного машинного коду набору $(x_1, x_2, \ldots, x_n)$, завжди закінчує роботу в кінцевому стані, лишаючи на стрічці основний машинний код числа 0, чи 1 в залежності від того, істинний предикат P на цьому наборі, чи хибний. Довести обчислюваність предиката $P(x_1, x_2, \ldots, x_n)$ означає, що потрібно створити машину Тьюрінга N, яка починаючи роботу зі стрічки з записом основного машинного коду набору $(x_1, x_2, \ldots, x_n)$, завжди закінчує роботу в в одному із кінцевих станів q', q'' в залежності від того, істинний предикат P на цьому наборі, чи хибний.

Отже маємо на стрічці основний машинний код набору $(x_1, x_2, \ldots, x_n,$ і маємо машину M, яка обчислює f. Потрібну нам машину N створюємо по кроках.

На першому кроці створюємо машину Тьрінга A з початковим станом q_{1a} і кінцевим станом q_{0a}, яка перезаписує основний код набору $(x_1, x_2, \ldots, x_n)$ на дві гратки S_1, S_2 з кроком 2, як в прикладі 1.8

На другому кроці за машиною M будуємо машину M_1 з початковим станом q_{1m} і кінцевим станом q_{0m}, яка на гратці S_2 обчислює функцію f.

На третьому кроці створюємо МТ B з початковим станом q_{1b} і кінцевими станами q_{01b}, q_{02b}, яка обчислює предикат $x = 0$. Кінцевий стан q_{01b} повідомляє про те, що $x = 0$ (предикат хибний), а кінцевий стан q_{02b} повідомляє про те, що $x \neq 0$ (предикат істинний, $x = 1$).

На четвертому кроці створюємо машину C з початковим станом q_1 і кінцевим станом q_0, яка переміщає голівка із гратки S_2 на гратку S_1 і відшукує на гратці S_1 ліву одиницю квазіосновного коду.

На п'ятому кроці створюємо машину D з початковим станом q_{1d} і кінцевим станом q_{0d}, яка перетворює квазіосновний код набору $(x_1, x_2, \ldots, x_n)$

на гратці S_1 в основний код на стрічці, як в прикладі 1.9, знищуючи при цьому записи на гратці S_2, і переміщає голівку до першої одиниці основного коду.

На шостому кроці створюємо композицію $E = AM_1B$ машин A, M_1, B за станами $q_{0a}, q_{1m}\ q_{0m}, q_{1c}$ з початковим станом q_{1a} і кінцевими станами q_{01b}, q_{02b}.

На сьомому кроці створюємо композицію $F = CD$ за станами q_{0c}, q_{1d}.

На восьмому кроці готуємо розгалуження машини E (у неї два кінцевих стани q_{01b}, q_{02b}) за допомогою машини F. Для цього дублюємо машину F — переписуємо двічі всі стани машини F (відповідно, перепишуться всі команди) один раз з додаванням індекса 1, а другий з додаванням індекса 2. Команди, в яких стани одержані додаванням індекса 1, утворюють машину F_1, а команди, в яких стани одержані додаванням індекса 2, утворюють машину F_2. Машини F_1, F_2 працюють однакова, так же як і машина F, але однакових станів у них немає. Початковим станом машини F_1 буде q_{1c1}, кінцевим станом машини F_1 буде q_{0d1}, початковим станом машини F_2 буде q_{1c2}, а кінцевим станом машини F_2 буде q_{0d2}.

На дев'ятому кроці будуємо кінцевий результата — машину N, як розгалуження машини E за допомогою машини F_1 (за станами q_{01b}, q_{1c1}) та машини F_2 (за станами q_{02b}, q_{1c2}). Завершення роботи машини N в стані q_{0d2} сигналізує про те, що предикат P на наборі $(x_1, x_2, \ldots, x_n)$ істинний, а завершення роботи в стані q_{0d1} сигналізує про те, що предикат P на наборі $(x_1, x_2, \ldots, x_n)$ хибний.

Обчислюваність предиката P доведена.

Припустимо тепер, що предикат P обчислюваний. Отже існує машина Тьюрінга M, яка, починаючи роботу з початкового стану q_1 із стрічкою, на якій записаний основний код набору $(x_1, x_2, \ldots, x_n)$, закінчує роботу в кінцевому стані q_{01}, якщо предикат на цьому наборі істинний, і в кінцевому стані q_{02}, якщо предикат на цьому наборі хибний. В обох випадках в кінці роботи машини голівка дивиться на

першу одиницю основного коду.

Позначимо через A, B дві копії (з різними наборами станів) машини, яка обчислює нуль, через C машину, яка обчислює $x+1$, і через D композицію BC. Тоді розгалуження машини M за допомогою машин B D буде машиною яка обчислює характеристичну функцію предиката P. Теорема доведеня повністю.

■

1.7 Універсальна машина Тьюрінга.

Програму машини Тьюрінга M можна записати за допомогою нулів та одиничок — хоча б використовуючи азбуку Морзе. Цей запис можна розташувати на гратці з кроком 2. На другій гратці можна записати дані, для роботи з якими створена ця машина M. Тепер можна створити нову машину U, яка за цими записами на стрічці виконує роботу машини M.

Така машина U називається універсальною машиною Тьюрінга. Програма універсальної МТ не складна, її можна знайти в інтернеті, але поширювані варіанти такої програми використовують досить багатий вхідний алфавіт. З використанням вкрай бідного алфавіту, що складається із 0 та 1, програма стає помітно громіздкою і наводити їїне будемо. Отже наступну теоремулишимо без доведення.

Теорема 1.5 *Існує універсальна машина Тьюрінга, тобто така машина U, яка для будь-якої машини M, і для будь-яких допустимих (для машини M) записів на стрічці, виконує роботу машини M.*

Ця теорема використовується при дослідженнях можливостей машин Тьюрінга.

2 Рекурсивні функції

Підхід Тьюрінга ми уже розібрали. Переходимо до іншого підходу.

2.1 Визначення частково рекурсивних, загально рекурсивних та примітивно рекурсивних функцій. Теза Чорча.

Частково рекурсивні функції визначаються індуктивно — вони будуються по кроках, 1-й, 2-й і т.д. На першому кроці вводяться три найпростіші частково рекурсивні функції, а на кожному наступному кроці із уже побудованих функцій будуються нові за допомогою трьох так званих операторів.

2.1.1 Три найпростіші обчислювані функції

Перша найпростіша функція $s(x) = x + 1$ — знаходження наступного числа або "плюс один". Це функція від однієї змінної.

Приклад 2.1

$$s(5) = 6, s(0) = 1.$$

Друга найпростіша функція — тотожний нуль. Це нульмісна функція, або виділений елемент в $\mathbb{N} \cup \{0\}$.

І третя найпростіша функція — проектування[3]

$$I_m^n(x_1, x_2, \ldots, x_n) = x_m, \quad 1 \leq m \leq n.$$

Ця функція від кількох змінних, вона вибирає один аргумент із списка. Тут можна вважати, що маємо нескінченно багато функцій (маємо два параметри — із скількох аргументів вибираємо, і який саме аргумент.

Задавши ці два параметри одержуємо одну функцію).

Приклад 2.2

$$I_3^5(x_1, x_2, x_3, x_4, x_5) = x_3, \quad I_2^5(2, 3, 0, 7, 1) = 3, \quad I_1^1(y) = y, \quad I_2^2(7, 0) = 0.$$

[3]Її називають також функцією вибору, селективною функцією

Інтуїтивно, згадані три функції є обчислюваними. А формально ці три функції є обчислюваними за означенням. Їх обчислюваність є аксіомою, яка не обговорюється.

Звернемо увагу, що всі три найпростіші функції є всюди визначені, ми їх можемо підрахувати при будь-яких наченнях аргументів.

2.1.2 Три оператори

Правила побудови одних функції із інших називають операторами.

Перший оператор — оператор суперпозиції.

Оператор C суперпозиції можна застосувати до $n+1-$єї функції $f, g_i (i = 1, 2, \dots n)$ і одержати функцію $h = C(f, g_1, g_2, \dots, g_n)$ наступним чином: якщо маємо

$$f(y_1, y_2, \dots, y_n), \quad g_i(x_1, x_2, \dots, x_m), \quad (i = 1, 2, \dots n),$$

то

$$h(x_1, x_2, \dots, x_m) = f(g_1(x_1, x_2, \dots, x_m), g_2(x_1, x_2, \dots, x_m), \dots, g_n(x_1, x_2, \dots, x_m)).$$

Тут обов'язково перша функція має n аргументів, за кількістю решти функцій. Отже коли $n = 2$, то перша функція $f(x_1, x_2)$ повинна мати 2 аргументи, а дві функції g_1, g_2 можуть бути від будь-якої кількості будь-яких аргументів. Щоб вони мали один і той же набір аргументів, можна вводити фіктивні змінні.

Приклад 2.3 *Візьмемо*

$$f(x_1, x_2) = 2x_1 + 3x_2, \qquad g_1 = 2^{x_1} + y, \qquad g_2 = u^3 + v^4 + 5.$$

Тоді

$$C(f, g_1, g_2) = h(x_1, y, u, v) = 2(2^{x_1} + y) + 3(u^3 + v^4 + 5).$$

Знову звернемо увагу, що коли оператор суперпозиції застосовується до всюди визначених функцій, то він дає всюди визначену функцію. В інших розділах математики, замість операторa суперпозиції

говорять про композицію функцій, про складні функції, функції від функцій.

Число n може дорівнювати 0, тоді перша функція f є сталою, функцією від порожньої множини змінних

Оскільки функція 0 має нуль аргументів, то $C(0) = 0$.

В курсі математичного аналізу розглядають функцію $0(x)$ — її графіком є вісь OX, функцію $0(x, y)$ — вона задає площину OXY. В такому підході може виникнути неприємна ситуація, коли замість змінних у функцію $0(x, y)$ потрібно підставити всюди невизначені функції. В нашому підході потрібно слідкувати за кількістю змінних у функції 0. Якщо це найпростіша функція, то вона не має аргументів. Але така функція може бути одержана як суперпозиція кількох функцій і тоді вона може мати будь-яку кількість аргументів.

Приклад 2.4 *Нехай* $n = 2$, $f(x, y) = 2x + 3y$, $g_1(x_1) = 2x_1, g_2(x_1) = x_1 + 2$. *Знайти* $h = C(f, g_1, g_2)$

$$h(x_1) = 2g_1 + 3g_2 = 4x_1 + 3(x_1 + 2) = 7x_1 + 6.$$

Приклад 2.5 *Нехай* $n = 0$, $f = 7$. *Знайти* $h = C(f)$

$$h = 7.$$

Приклад 2.6 *Нехай* $n = 1$, $f(x) = 2x$, $g_1(x_1, x_2) = 2x_1 + 3x_2$. *Знайти* $h = C(f, g_1)$.

$$h(x_1, x_2) = 2g_1 = 4x_1 + 6x_2.$$

Переходимо до наступного оператора — оператора примітивної рекурсії Пр.

Оператор примітивної рекурсії[4] (позначаємо його Пр) можна застосувати до двох функцій $\phi(x_1, x_2, \ldots x_n), \psi(x_1, x_2, \ldots x_n, x_{n+1}, x_{n+2})$ і одержати функцію $f(x_1, x_2, \ldots x_n, x_{n+1})$ за допомогою двох правил. Перше правило

$$f((x_1, x_2, \ldots x_n, 0) = \phi(x_1, x_2, \ldots x_n) \tag{6}$$

і друге правило

$$f((x_1, x_2, \ldots x_n, y+1) = \psi(x_1, x_2, \ldots x_n, y, f((x_1, x_2, \ldots x_n, y)) \tag{7}$$

Звернемо увагу, що вище n означало кількість змінних. Ця кількість може бути нулем. Отже функція ϕ може бути нульмісною. Тоді функція ψ має бути двомісною (змінні можуть бути фіктивні[5]), і за допомогою оператора Пр одержуємо 1-місну функцію.

Слово рекурсія означає рух назад, а індукція означає рух вперед. Рекурсія і індукція тісно пов'язані між собою. Так, коли ми кажемо, що для обчислення $n!$ нам потрібно обчислити спочату $(n-1)!$, то ми корстуємося рекурсією. Коли ж ми кажемо, що після обчислення $(n-1)!$ ми можемо перейти до обчислення $n!$, то ми маємо індукцію. Рівність (6) можна розглядати як базу індуктивного означення функції f, а рівність (7) можна розглядати як індуктивний перехід в індуктивному означенні цієї функції. Тонкощами розрізнення індуції та рекурсії можна нехтувати, але в програмуванні широко використовується рекурсія, а в логіці індукція.

Приклад 2.7 *Застосуємо оператор примітивної рекурсії до функцій*

$$\phi(x_1) = x_1 = I_1^1(x_1), \qquad \psi(x_1, x_2, x_3) = x_3 + 1 = s(x_3).$$

[4]Оператор примітивною рекурсії називають також схемою примітивної рекурсії, подібно до того, як ми виписували схеми аксіом в численні висловлювань.

[5]формальне означення фіктивних змінних обговорювалися в булевих функціях. Тут користуємося нестрогим роз'ясненням: змінна фіктивна, якщо від неї функція не залежить

Функція ϕ від однієї змінної, функція ψ від трьох змінних, отже будуємо функцію $f(x_1, x_2)$ від двох змінних:

$$f(x_1, 0) = x_1, \quad f(x_1, 1) = f(x_1, 0) + 1 = x_1 + 1,$$

$$f(x_1, 2) = f(x_1, 1) + 1 = x_1 + 2, \quad f(x_1, 3) = f(x_1, 2) + 1 = x_1 + 3, \ldots$$

Вгадуємо, $f(x_1, x_2) = x_1 + x_2$. Вгадане можна обгрунтувати методом матіндукції. Таким чином

$$x_1 + x_2 = \Pr(I_1^1(x_1), s(I_1^3(x_1, x_2, x_3))). \tag{8}$$

Приклад 2.8 *За допомогою оператора примітивної рекурсії можна визначити функції* $0(x), 0(x, y), \ldots$.

Так якщо взяти $\phi = 0$ — нульмісна функція, і $\psi(x_1, x_2) = x_1$, то результатом роботи оператора примітивної рекурсії буде функція $0(x)$.

Приклад 2.9 *Сталі функції від однієї змінної (отже ця змінна є фіктивною) визначаються правилом:* $f(y + 1) = f(y)$. *Тому для визначення сталої функції можна брати функцією* ϕ *кількаразову суперпозицію функці* s *і 0. А функцією* $\psi(x_1, x_2)$ *можна брати* $\psi(x_1, x_2) = x_2$. *Так, якщо застосувати оператор примітивної рекурсії до функцій*

$$\phi = s(s(s(s(0)))), \qquad \psi(x_1, x_2) = x_2,$$

то одержимо сталу функцію $f(x) = 4$.

Приклад 2.10 *Застосуємо оператор примітивної рекурсії до функцій*

$$\phi(x_1) = 0(x_1), \qquad \psi(x_1, x_2, x_3) = x_1 + x_2.$$

Функція ϕ від однієї змінної, функція ψ від трьох змінних, отже будуємо функцію $f(x_1, x_2)$ від двох змінних:

$$f(x_1, 0) = 0, \quad f(x_1, 1) = 0 + f(x_1, 0) = x_1,$$

$$f(x_1, 2) = x_1 + f(x_1, 1) = 2x_1, \quad f(x_1, 3) = x_1 + f(x_1, 2) = 3x_1, \ldots$$

Вгадуємо, $f(x_1, x_2) = x_1 \cdot x_2$. Вгадане можна обгрунтувати методом матіндукції. Таким чином

$$x_1 \cdot x_2 = \Pr(0, x_1 + x_2). \tag{9}$$

Приклад 2.11 *Прямою перевіркою переконуємося, що застосувавши оператор примітивної рекурсії до функцій* $\phi(x_1) = 0 = 0(x_1), \quad \psi(x_1, x_2, x_3) = s(0)$, *одержимо функцію*

$$f_1(x) = \begin{cases} 0, & \text{якщо } x = 0, \\ 1, & \text{якщо } x \neq 0, \end{cases} \tag{10}$$

Приклад 2.12 *Прямою перевіркою переконуємося, що застосувавши оператор примітивної рекурсії до функцій* $\phi(x_1) = 1 = s(0(x_1)), \quad \psi(x_1, x_2, x_3) = 0$, *одержимо функцію*

$$f_2(x) = \begin{cases} 1, & \text{якщо } x = 0, \\ 0, & \text{якщо } x \neq 0, \end{cases} \tag{11}$$

Приклад 2.13 *Застосувавши оператор примітивної рекурсії до функції* $\phi(x_1) = 1 = s(0(x_1))$, *та функції* $\psi(x_1, x_2, x_3) = f_2(x_2)$, *що визначена формулою (11) одержимо функцію*

$$f_3(x) = \begin{cases} 1, & \text{якщо } x = 2k, \text{ для деякого } k = 0, 1, \ldots \\ 0, & \text{якщо } x = 2k + 1 \text{ для деякого } k = 0, 1, \ldots. \end{cases} \tag{12}$$

Приклад 2.14 *Застосувавши оператор примітивної рекурсії до функції* $\phi(x_1) = 1 = 0$, *та функції* $\psi(x_1, x_2, x_3) = f_2(x_2)$, *що визначена формулою (11) одержимо функцію*

$$f_4(x) = \begin{cases} 0, & \text{якщо } x = 2k, \text{ для деякого } k = 0, 1, \ldots \\ 1, & \text{якщо } x = 2k + 1 \text{ для деякого } k = 0, 1, \ldots. \end{cases} \tag{13}$$

І, нарешті, переходимо до третього, останнього оператора, оператора мінімізації.

Оператор мінімізації застосовують до однієї функції $g(x_1, x_2, \ldots x_n, x_{n+1})$ і одержують функцію $f(x_1, x_2, \ldots x_n, x_{n+1})$. Значення y цієї функції $y = f(x_1, x_2, \ldots, x_n, x_{n+1})$ визначається як найменший розв'язок рівняння

$$g(x_1, x_2, \ldots x_n, y) = x_{n+1}$$

Розв'язок шукаємо підставляючи по черзі замість y числа $0, 1, 2, \ldots$. Перший знайдений розв'язок і буде потрібним. Застосування цього оператора повинно нагадувати знаходження оберненої функції.

Приклад 2.15 *Візьмемо $n = 1$ і застосуємо оператор мінімізації до $g(x) = x + 1$.*

Знаходимо значення $y = f(0)$ $g(y) = 0$, тобто $x + 1 = 0$. Оскільки розв'язку це рівняння не має, то значення $f(0)$ немає, функція f не изначена в точці 0. А для $x \geq 1$ $f(x) = x - 1$.

Таким чином, за допомогою оператора мінімізації ми одержали функцію

$$x - 1 = \begin{cases} \text{не визначено}, & \text{якщо } x = 0, \\ x - 1, & \text{якщо } x \neq 0. \end{cases} \tag{14}$$

Повернемося до означення частково рекурсивних функцій, яке ми окреслили на початку розділу.

Визначення 2.1 *Частково рекурсивні функції будуються по кроках. На першому кроці будуються найпростіші функції — нуль, обчислення наступного натурального числа, та функція вибору одного аргумента із набору. Якщо на $n-$ му кроці маємо побудовані частково рекурсивні функції, то на $n + 1-$ му кроці до уже побудованих додаються функції, що одержані із уже побудованих за допомогою операторів суперпозиції, примітивної рекурсії та мінімізації. Інших частково рекурсивних функцій (які неможливо так одержати) немає.*

Приклад 2.16 *Застосувавши оператор примітивної рекурсії до функції* $\phi(x_1) = 0$, *та функції* $\psi(x_1, x_2, x_3) = x - 1$,

$$x\dot{-}1 = \begin{cases} 0, & \text{якщо } x = 0, \\ x - 1, & \text{якщо } x \neq 0. \end{cases} \tag{15}$$

Приклад 2.17 *Якщо функція* $f(x)$ *побудована за функцією* $g(x) = 2x$, *то* $f(x) = x/2$ *для парних* x *і* $f(x)$ *не визначена для непарних. Отже, якщо функція* $g(x)$ *є частково рекурсивною, то і* $f(x)$ *є частково рекурсивною.*

Нехай функції f_3, f_4 визначені формулами (12)(13). Тоді

$$f_{ev}(x) = f_3(x) \cdot x = \begin{cases} x, & \text{якщо } x = 2k, \text{ для деякого } k = 0, 1, \ldots \\ 0, & \text{якщо } x = 2k + 1 \text{ для деякого } k = 0, 1, \ldots. \end{cases}$$

і

$$f_{odd}(x) = f_4(x) \cdot x = \begin{cases} 0, & \text{якщо } x = 2k, \text{ для деякого } k = 0, 1, \ldots \\ x, & \text{якщо } x = 2k + 1 \text{ для деякого } k = 0, 1, \ldots. \end{cases}$$

Із цих функцій та функції $x_1 + x_2$ за допомогою оператора суперпозиції одержуємо функцію

$$\left[\frac{x}{2}\right] = \frac{f_{ev}(x)}{2} + \frac{f_{odd}(x)\dot{-}1}{2}. \tag{16}$$

Приклад 2.18 *Розглянемо сталу функцію* $g(x) = 5$. *Щоб знайти значення* $y = f(0)$ *ми повинны розв'язати рівняння* $g(x) = 0$, *тобто 5=0. Це рівняння не має розв'язку, отже* $f(0)$ *не визначено. Також не визначено* $f(1), f(2), f(3), f(4), f(x)$ *коли* $x > 5$. *Знайдемо* $f(5)$. *Для цього розв'язуємо рівняння* $5 = 5$. *Ми маємо вельми точну вказівку, як саме шукати розв'язок — потрібно по черзі підставляти замість змінної числа 0, 1, 2, ... Оскільки 5=5 завжди, то 5=5 і при першій же перевірці — коли змінна дорівнює 0. Отже* $y = 0$ *і* $f(5) = 0$.

Приклади показують, що оператор мінімізації суттєво відрізняється від перших двох — суперпозиції та примітивної рекурсії тим, що він може давати часткову, не всюди визначену функцію.

Визначення 2.2 *Якщо при побудові функції дозволене використання всіх трьох операторів, але функцію одержали всюду визначену, то цю функцію назають загальнорекурсивною. Якщо ж про область визначення нічого невідомо, або відомо, що функція не всюди визначена, то ця функція частково рекурсивна. Якщо при побудові функції використовувалося лише два оператори — суперпозиції і і примітивної рекурсії, то таку функцію називають примітивно рекурсивною.*

Оскільки найпростіші функції всюди визначені, і оператори примітивної рекурсії та суперпозиції при застосувані їх до всюди визначених функцій дають всюди визначені функції, то примітивно рекурсивні функції завжди всюди визначені.

В мовній практиці тонкощами слововжитку часто нехтують і кажуть про рекурсивні функції — а про які саме, або видно із мовного оточення (контексту) або це не має значення.

Всі функції, що побудовані в прикладах розділу, є рекурсивними. Але функція xy є примітивно рекрсивною, функція $x - 1$ є частково рекурсивною, а функція $x \dot{-} 1$ є загально рекурсивною функцією.

> **Теза Чьорча:** Всі обчислювані в інтуїтивному розумінні функції є частково рекурсивними.

Будемо користуватися тим, що всі функції на розширеному натуральному ряді, які можна задати арифметичними виразами в шкільному розумінні, є рекурсивними.

Після того, як ми вказали два підходи до формалізації інтуїтивної обчислюваності, і вказали дві тези — Тьюрінкга і Чорча, можна говорити просто про обчислювані функції — а чи вони обчислювані на машині Тюрінкга, чи вони є частково рекурсивними функціями, чи

для них є по іншому вказаний алгоритм обчислення, — все це не має принципового значення.

2.2 Перераховні множини

Визначення 2.3 *Область значень обчислюваної функції називається перераховною множиною. Якщо перераховна множина $M \subseteq \overline{\mathbb{N}}$ є множиною всіх значень обчислюваної функції f, то кажуть, що M перераховується функцією f.*

Приклад 2.19 *Множина всіх натуральних чисел є перераховною множиною, тому що це область значень обчислюваної функції $s(x) = x + 1$.*

Приклад 2.20 *Весь розширений натуральний ряд є перераховною множиною, оскільки це область значень функції $f(x) = x = I_1^1(x)$.*

Приклад 2.21 *В попередньому розділі ми розглянули частково рекурсивну функцію (див. приклад 2.18), у якої область визначення складається лише із 5, а область значень лише із 0. Отже множина, що складається лише із 0, є перераховною.*

Приклад 2.22 *Наявність загальнорекурсивних функцій $f_{ev}, f_{odd}, (f_{odd} - 1) + 1$ доводить, що множина парних чисел і множина непарних чисель з нулем є перераховними. А наявність частково рекурсивної функції $(f_{odd} - 1) + 1$ доводить, що множина непарних чисел є перераховною.*

Доведемо, що перетин і об'єднання двох перераховних множин є перераховними множинами.

Теорема 2.1 *Перетин двох перераховних множини є перераховною множиною.*

Доведення. Нехай множини $A, B \subseteq \overline{\mathbb{N}}$ є перераховними і перераховуються вони функціями $u(x), v(x)$ відповідно, тобто A є множиною значень функції $u(x)$ а B є множиною значень функції $v(x)$. Позачимо через $v^{-1}(x)$ функцію, яка одержується із функції $v(x)$ застосуванням оператора мінімізації. Таким чином $v(v^{-1}(x) = x$, якщо x належить області значень функції $v(x)$. І функція $v^{-1}(x)$ не визначена в точці x, якщо x не належить області значень функції $v(x)$.

Розглянемо функцію

$$w(x) = v(v^{-1}(u(x))).$$

Вона є обчислюваною, оскільки є композицією обчислюваних функцій.

Якщо $y \in A \cap B$, то $v(v^{-1}(y) = y$, (за визначенням функції $v^{-1}(x)$) і $y = u(x)$ для деякого x (за визначенням області значень функції u). І для так вибраного x буде

$$w(x) = v(v^{-1}(u(x))) = v(v^{-1}(y) = y.$$

Ми довели, що коли $y \in A \cap B$, то y належить області значень функції w.

Якщо число $w(x)$ визначене, то

$$v(v^{-1}(u(x))) = u(x). \tag{17}$$

Ліва частина рівності (17) показує, що область значень функції w міститься в B. А права частина рівності (17) показує, що область значень функції w міститься в A. Отже область значень функції w міститься в перетині $A \cap B$.

Доведення закінчене.

■

Доведемо, що об'єднання перераховних множин є перераховною множиною. З огляду на важливість твердження сформулюємо його у вигляді теореми.

Теорема 2.2 *Об'єднання двох перерховних множин є перераховна множина.*

Доведення. Нехай множини $A, B \subseteq \overline{\mathbb{N}}$ є перераховними і перераховуються вони функціями $u(x), v(x)$ відповідно, тобто A є множиною значень функції $u(x)$ а B є множиною значень функції $v(x)$.

Користуємося тим, що функції

$$f_{odd}, \quad f_{ev}, \quad \frac{x}{2}, \quad x+1, \quad x-1, \quad x\dot{-}1$$

є рекурсивними — це доведене вище. Для більшої наочності будемо використовувати табличне задання функції — двома рядками: у верхньому рядку пишемо значення аргумента, а в нижньому пишемо значення функції.

Застосовуємо оператор примітивної рекурсії до функцій $\phi(x_1) = 0 = 0(x_1), \qquad \psi(x_1, x_2, x_3) = u(x_3)$. Одержану функцію позначимо через u_1. Таким чином

$$u_1(n) = \begin{cases} 0, & \text{якщо } n = 0, \\ u(n-1), & \text{якщо } n \neq 0 \end{cases} \qquad u_1 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & \dots \\ 0 & u(0) & u(1) & u(2) & u(3) & u(4) & \dots \end{pmatrix}$$

Суперпозицій функцій $\frac{x}{2}, f_{ev}(x)$ буде функція

$$\frac{f_{ev}(x)}{2} = \begin{cases} x, & \text{якщо } x = 2k, \text{ для деякого } k = 0, 1, \dots \\ 0, & \text{якщо } x = 2k+1 \text{ для деякого } k = 0, 1, \dots . \end{cases} \quad \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & \dots \\ 0 & 0 & 1 & 0 & 2 & 0 & \dots \end{pmatrix}$$

Створюємо суперпозицію

$$u_1\left(\frac{f_{ev}(x)}{2}\right) = \begin{cases} u_1(x), & \text{якщо } x = 2k, \text{ для деякого } k = 0, 1, \dots \\ u_1(0), & \text{якщо } x = 2k+1 \text{ для деякого } k = 0, 1, \dots . \end{cases}$$

В табличному записі маємо

$$u_1\left(\frac{f_{ev}}{2}\right) = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & \dots \\ u_1(0) & u_1(0) & u_1(1) & u_1(0) & u_1(2) & u_1(0) & \dots \end{pmatrix} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & u(0) & 0 & u(1) & 0 \end{pmatrix}$$

Суперпозицією одержаної функції і функції $x+2$ буде функція

$$u_2 = u_1\left(\frac{f_{ev}(x+2)}{2}\right) = \begin{cases} u(k), & \text{якщо } x = 2k, \text{ для деякого } k = 0,1,\ldots \\ 0, & \text{якщо } x = 2k+1 \text{ для деякого } k = 0,1,\ldots. \end{cases}$$

Подібним чином будуємо функцію

$$v_1(n) = \begin{cases} 0, & \text{якщо } n = 0, \\ v(n-1), & \text{якщо } n \neq 0 \end{cases}$$

і функцію

$$v_2 = v_1\left(\frac{f_{ev}(x+1)}{2}\right) = \begin{cases} 0, & \text{якщо } x = 2k, \text{ для деякого } k = 0,1,\ldots \\ v(k), & \text{якщо } x = 2k+1 \text{ для деякого } k = 0,1,\ldots. \end{cases}$$

Суперпозицією функцій $x_1 + x_2$ і u_2, v_2 буде функція

$$u_2(n) + v_2(n) = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & \ldots \\ u(0) & v(0) & u(1) & v(1) & u(2) & v(2) & \ldots \end{pmatrix}$$

областю значень якої є $A \cup B$.

Теоретико-множинна різниця перераховних множин не обов'зково перераховна множина.

■

2.3 Обчислювані предикати.

Визначення 2.4 *Предикат $P(x_1, x_2, \ldots, x_n)$ з вільними змінними $x_1, x_2, \ldots, x_n$ називають обчислюваним, коли обчислюваною є характеристична функція області істиності цього предиката*

$$f(x_1, x_2, \ldots, x_n) = \begin{cases} 1, & \textit{якщо } P(x_1, x_2, \ldots, x_n) = 1, \\ 0, & \textit{якщо } P(x_1, x_2, \ldots, x_n) = 0. \end{cases}$$

Кажуть, що предикат $P(x_1, x_2, \ldots, x_n)$ обчислюється функцією f.

Наведемо приклади.

Приклад 2.23 *Одномісний предикат $z = 0$ обчилюваний, він обчислюється функцією*

$$f(z) = \begin{cases} 1, & \text{якщо } z = 0, \\ 0, & \text{якщо } z \neq 0. \end{cases}$$

Приклад 2.24 *Двомісний предикат $x = y$ обчилюваний, він обчислюється функцією*

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

де функцію f взяли із попереднього прикладу.

Приклад 2.25 *Одномісний предикат $\exists x(2x = y)$ обчислюваний, він обчислюється функцією $f_3(y)$ (див. формулу (12))*

Будемо вважати очевидним, що коли предикат P обчислюється функцією f, предикат Q обчислюється функцією g, то предикат $P \wedge Q$ обчислюється функцією $f \cdot g$, а предикат $P \vee Q$ обчслюється функцією $f + g - fg$.

Визначення 2.5 *Множина M називається обчислюваною, коли обчислюваним є предикат $x \in M$.*

2.4 Конструктивний напрям в математиці

Вище відмічено, що існують необчислювані функції, предикати, множини. так ми уявляємо, що в кожній непорожній множині можна вибрати один елемент, але алгоритма, як вибирати цей елемент, немає. Виникає питання, а які твердження в математиці можна обгрунтувати, використовуючи лише ті засоби, які забезпечуються відповідними алгоритмами. Напрям математики, в якому всі теореми проходять

тестування цим питанням, називається конструктивним. Для математика, який сповідує конструктивний напрям, функція

$$f(0) = \begin{cases} 1, & \text{якщо } z = 0, \\ 0, & \text{якщо } z \neq 0. \end{cases}$$

на множині дійсних чисел не існує, тому що немає алгоритма, який би дозволив перевірити — задане дійсне число є нулем чи ні.

Число $\sqrt{2}$ існує, тому що можна вказати алгоритм послідовної побудови цифр в його десятковому записі. Взагалі, дійсні числа з конструктивної точки зору це алгоритми, що дозволяють знайти будь-яку (скінченну!) множину цифр в десятковому записі.

Увесь конструктивний напрям не однорідний — він розділяється на дві течії. Одна вимагає точного формулювання поняття алгоритм. Це так звана конструктивна математика. Основоположниками напрямку можна вважати творців частково рекурсивних функцій.

Друга течія користується інтуїтивним розумінням поняття алгоритм. Це так званий інтуїціонізм[6]. Відлік часу життя інтуїціонізму починається з 1907 року, з першої роботи Брауера (Brouer)

Література

[1] Хромой Я.В. Збірник вправ і задач з математичної логіки. — Київ:Вища школа, 1978.

[2] Лавров И.А. Максимова Л.Л. Задачи по теории множеств, математической логике и теории алгоритмов. — М:Наука, 1984

[3] Гаврилов Г.П. Сапоженко А.А. Сборник задач о дискретной математике. — М:Наука, 1977

[6] Не плутати з філософськиим напрямком — інтуїтивізмом

Покажчик