Факультет комп'ютерних наук та кібернетики
Київського національного університету імені Тараса Шевченка

В.В. Зубенко
О.В. Шишацька
Ф.А. Асроров

АЛГОРИТМІЧНІ ПРОЦЕДУРИ

Навчальний посібник

Київ-2023

УДК 510.51(075.8)
ББК 22.12я73
Ш66

Рекомендовано вченою радою факультету комп'ютерних наук та кібернетики КНУТШ
(протокол № 9 від 21 березня 2023 року)

Рецензенти:

С.С. Шкільняк – проф., д-р фіз.-мат. наук,
П.П. Кулябко - доцент, канд. фіз.-мат.наук

Зубенко В.В., Шишацька О.В., Асроров Ф.А.

Ш66 **Алгоритмічні процедури:** навчальний посібник. К.: НУБіП України, 2023. 56 с.

Викладено основи процедурної платформи для теорії алгоритмів. Досліджено загальні властивості алгоритмічних процедур і місце традиційних моделей алгоритмів у цій платформі. Запропоновано вправи для самостійного розв'язку. Посібник склали матеріали курсів лекцій та семінарських занять з математичної логіки та теорії алгоритмів, які читають автори на факультеті комп'ютерних наук та кібернетики КНУ ім. Тараса Шевченка.

Для студентів та магістрів освітніх програм «Інформатика», «Прикладна математика» та «Системний аналіз».

ISBN 978-617-7878-13-0

ЗМІСТ

Вступ

Теорія алгоритмів – наука про алгоритми та їхні загальні властивості. Вона досліджує формальні моделі алгоритмів і алгоритмічно обчислюваних функцій і часто методично є частиною курсів з математичної логіки, в недрах якої і виникла у першій половині XX ст.

Як відомо, термін алгоритм походить від імені перського математика IX ст. Аль-Хорезмі, який вперше в систематичному вигляді описав арифметичні операції в 10-й позиційній системі і сформулював правила для їх виконання в тому вигляді, в якому ми з вами сьогодні ними користуємося і називаємо алгоритмами додавання, множення тощо.

Пізніше цим терміном стали називати усі чітко сформульовані правила для отримання потрібних результатів, виходячи з конкретних вхідних даних з певної сукупності, за допомогою строго визначеної послідовності дії. Такі послідовності отримали назву *обчислень* за алгоритмом. Іноді обчисленням називають не саму послідовність дій, а послідовність результатів цих дій на проміжних даних.

Насправді, алгоритми є частковим випадком більш широкого і давнього поняття – процедури, яке походить від французьких слів procedure (процедура, операція, процес) та procedere (діяти в часі, рухатися вперед). Кому не відомі медичні, юридичні, танцювальні, музичні, ігрові чи кулінарні процедури[1] або більш сучасні процедури мов програмування.

Процедури, як і алгоритми, є інструкціями для виконання дій з певною метою. При цьому вони можуть явно і не повертати конкретний результат. У цьому випадку сенсом роботи процедури є сам процес обчислення, як це має місце, наприклад, при виконанні танцю чи музичного твору.

Основна відмінність між процедурами та алгоритмами полягає у вимогах до способу їхнього подання. На відміну від процедур, алгоритми потребують точного (часто формального чи формалізованого[2]), обов'язково

[1] Останні у вигляді рецептів страв.

[2] Формалізована мова - це фрагмент природної мови з однозначною семантикою конструкцій і елементами формальної мови.

вичерпного і скінченного за розміром, як свого опису, так і тих об’єктів, з якими вони мають справу.

Прикладами такої вичерпності і точності опису можуть слугувати згадані вище арифметичні правила чи запис шахової партії. На відміну від алгоритмів, процедури, як правило, описуються природною мовою, конструкції якої можуть допускати різну інтерпретацію і бути не повними. Наприклад, у різних господарок за одним і тим же рецептом борщу результати на виході (на смак) будуть, як правило, різними, а от річницю держави України у 2091 році за алгоритмом віднімання 2091 - 1991 ті ж господарки отримають однакову – 100.

Якщо поглянути на процедури історично, то потреба в діях та їхньому описі відображена вже в самих ранніх природніх мовах. Найпростіші з дій - будемо називати їх далі *елементарними* - представлені в них дієсловами, які можуть супроводжуватися супутніми конструкціями, що уточнюють дію. Наприклад, зробити крок, зробити крок вправо, зробити крок повільніше, тощо.

Більш складні дії складаються з сукупності елементарних дій, виконуваних в певній послідовності. Вони називаються *складеними.* Складена дія може й не закінчуватись, а продовжуватися нескінченно, наприклад, коли вона “зациклюється” як при наказі крокувати по колу.

З розвитком абстрактного мислення, а з ним і природних та штучних мов, з’явилася можливість описувати дії, не прив’язуючись жорстко до контексту їхнього виконання, а навпаки - абстрагуючись від нього. Насамперед це стосується початкових умов виконання процедури.

Для подання вхідних даних стали використовувати тільки їхні імена (параметри), пов’язані з певною сукупністю можливих значень, а не з конкретними значеннями. Такі процедури описують цілий клас однотипних дій, що відповідає усім можливим варіантам значень параметрів. Щоб виконати дію, яку описує параметризована процедура, потрібно актуалізувати її параметри, тобто пов’язати з ними конкретні значення. Параметризуватися можуть і цілі внутрішні фрагменти процедур з наступною їхньою актуалізацією в процесі виконання. До речі, параметризація по суті вже має місце у згадуваних вище арифметичних

правилах Аль-Хорезмі. Кожне з них має два вхідні параметри: ”перше число” та ”друге число”, які представляють десяткові числа.

Сформулюємо мінімальні вимоги до опису конкретної процедури та її дії.

1) Має бути описане коло S об’єктів – станів, до яких застосовна процедура і якими вона маніпулює. Окремо виділена, у разі необхідності, підмножина можливих початкових станів S_0.

2) Має бути описана сукупність Δ всіх ементарних дій $f : S \to S$ процедури на станах.

3) Має бути описаний порядок виконання елементарних дій за процедурою для всіх варіантів вхідних даних. Це означає наявність функції керування процесом обчислення (не обов’язково однозначної!) $\bar{\delta} : S_0 \times \Delta^+ \to \Delta$, яка за початковим станом і будь-якою виконаною послідовністю дій повертає перелік можливих елементарних дій процедури на наступному кроці.

4) Має бути заданим механізм завершення роботи процедури і визначення результату. Будемо вважати умовою завершення процесу обчислення порожнє значення функції керування після виконання крайнього кроку у ньому. Як свідчить досвід використання алгоритмів та існуючих їхніх моделей такого механізму достатньо.

5) Якщо в процедурі є параметри, то має бути окреслено коло їхніх можливих значень.

Як правило, процедури формулюються в природній мові чи у деякому формалізованому її фрагменті. Останнє стосується в основному пп. 2)-3). Якщо тепер до цих п’ятьох вимог додати вимогу 6) про те, щоб подання як самої процедури, так і усіх об’єктів, з якими вона має справу, було вичерпним (повним) і скінченним (фінітним), то отримаємо *природничо-наукове* чи, як ще кажуть, *інтуїтивне* визначення процедурного алгоритму.

Різницю між процедурою і процедурним алгоритмом ілюструє добре відома ще з середньої школи процедура поділу навпіл відрізка на площині за допомогою циркуля і лінійки. Якщо її обмежити тільки відрізками з цілочисловими чи раціональними координатами на площині, то вона буде і

алгоритмом, а якщо ж не обмежувати, то формально - ні, тому що ірраціональні числа, як правило, не мають скінченного подання.

Подивимось тепер як співвідносяться загальне поняття алгоритму і його процедурний аналог. Зазвичай, загальне поняття алгоритму пов'язують зі способом подання сукупності обчислювальних процесів для розв'язку задач певного класу [1,6]. Характеристичні вимоги до цього способу фактично збігаються з аналогічними вимогами до опису алгоритмічних процедур та їхніх обчислень за виключенням п.3), в якому йдеться про вибір чергової дії в обчисленні. У випадку процедурних алгоритмів сукупність цих можливих дій визначає спеціальна функція керування, а в загальних алгоритмах даний вибір поточної дії не уточнюється, а замість цього стверджується, що поточна дія (чи дії) на кожному кроці обчислення є строго визначеними і визначаються історією обчислення за алгоритмом, тобто за початковим станом і тими кроками, які було до цього зроблені (керованість діями). Більш детально про співвідношенням між інтуїтивними алгоритмами і процедурними буде йти мова у Розд.2 (Лема 2.1).

Інтуїтивного чи процедурного визначення алгоритму цілком достатньо, коли мова йде про окремо взятий алгоритми та проведення за ним обчислень. Але в середині XIX ст. виникли дві обставини, які докорінно змінили ситуацію і вимагали розгляду вже не окремих алгоритмів, а їхніх сукупностей і навіть всю їхню можливу сукупність.

В 1834-37 рр. англійський математик Ч.Бебідж запропонував проект Аналітичної машини, яка за задумом мала самостійно проводити числові обчислення за спеціальними програмами - алгоритмами, поданими у певній мові (мові програмування). Машина не була побудована, але вона привела до загальної ідеї автомата з програмним керуванням (АПК), яка незабаром знайшла своє теоретичне обґрунтування і практичну реалізацію. В моделях АПК мови програмування мають бути формальними для того, щоб їхні програми можна було однозначно трактувати і автоматично виконувати.

В зв'язку з автоматами з програмним керуванням природньо постає питання про масштаб їхніх обчислювальних можливостей. Очевидно, що цей масштаб визначається класом програм, які здатен виконувати конкретний АПК. Виникає, принаймні теоретично, і загальне питання про

існування універсальних АПК, спроможних реалізувати дії за будь-якими числовими алгоритмами. Але щоб підійти до відповіді на нього, потрібно, як мінімум, точно сформулювати, що таке алгоритм. Іншими словами, коли мова йде про конкретний алгоритм і його виконання, то спеціальних вимог до його подання немає, хіба що він має бути зрозумілим для виконавця. А от створити конкретну мову програмування для універсального числового АПК означає вже дати формальний опис усіх можливих числових алгоритмів.

Друга вагома обставина пов'язана зі створенням у XIX ст. формальних числень і аксіоматизацією математичних теорій, зокрема, арифметики (аксіоматика Пеано). З появою точного визначення арифметичної теореми та її доведення стало можливим не тільки запитати чи буде довільне арифметичне твердження теоремою чи ні, а й і чи існує алгоритм для відповіді на це питання.

На початку XX ст. в математиці накопичилася велика купа подібних відкритих проблем. Вони пов'язані з певним твердженням – усюди визначеним предикатом і пошуком алгоритму для обчислення його значень для кожного з варіантів значень змінних. Тому такі проблеми отримали назву *масових проблем*. Коли такий алгоритм існує, то говорять, що масова проблема *розв'язна*, а якщо – ні, то проблема називається *алгоритмічно нерозв'язною.*

Враховуючи, що частина з масових проблем були надскладними і перебували в статусі відкритих десятиліттями і навіть століттями, як та ж проблема Ферма чи гіпотеза Гольдбаха, то виникли серйозні підстави вважати, що такі проблеми є алгоритмічно нерозв'язними. Але щоб це математично довести, потрібно було перетворити алгоритм з природничо-наукового поняття в математичний об'єкт.

Таким чином, на початку XX ст. перед математичною спільнотою постала нагальна проблема математичного уточнення поняття алгоритму.[3] В

[3] Цікаво, що серед 23 знаменитих відкритих проблем, проголошених Д.Гільбертом в 1900 році як особливо важливими для подальшого розвитку математики, 10-та була конструктивною і стосувалася пошуку алгоритму для розв'язку діофантових рівнянь в цілих числах. Вочевидь, формулюючи дану масову проблему, сам Д.Гільберт не ставив під сумнів її розв'язність. Можливо, тому і не долучив до списку як одну з важливих проблем - математичне уточнення алгоритму. Але як з'ясувалося, вона алгоритмічно нерозв'язна і її розв'язок потребує такого уточнення. (Ю.Матиясевич, 1970).

30-х роках XX ст. ця проблема була розв'язана в роботах А.Чорча, А.Тьюрінга, Е.Поста, С.Кліні, які запропонували перші формальні моделі алгоритмів та алгоритмічно обчислювальних функцій і заклали тим самим підвалини сучасної теорії алгоритмів (ТА).

Зазначимо, що список моделей алгоритмів постійно поповнюється і налічує вже кілька десятків, а якщо враховувати й формальні моделі мов програмування, то їхнє число буде в рази більшим. Сама ж теорія алгоритмів давно вийшла за межі проблематики математичної логіки і стала окремою математичною дисципліною.

Метою даного посібника є ознайомлення студентів з процедурною платформою для теорії алгоритмів і вивчення деяких загальних властивостей алгоритмів на базі цієї платформи. Буде також зроблено огляд існуючих на сьогодні моделей алгоритмів в рамках процедурної платформи.

Перший розділ присвячений формальному уточненню поняття процедури, на якому базується процедурна платформа. Другий – конструктивним процедурам, які є – центральним поняттям у запропонованій платформі. У третьому розділі розглянуті деякі загальні властивості алгоритмів як процедур. Четвертий - присвячений огляду традиційних моделей алгоритмів і опису їхнього місця серед алгоритмічних процедур.

Всі розділи супроводжуються контрольними питаннями і вправами для самостійної роботи. Звернемо увагу на специфіку цитування літературних джерел в посібнику: в одному переліку джерел вони можуть розташовуються не за порядком їхнього розташування в списку Літератури, а за певним пріоритетом. Враховувалися різні фактори такі як першоджерело, доступність для студентів як учбового матеріалу та інші.

Контрольні запитання

1. Що таке процедура як природничо-наукове поняття?
2. Як ви розумієте фразу: «Процедури навколо нас» ?
3. Які основні атрибути процедур ?

4. Що таке функція керування процедури?
5. Що таке обчислення за процедурою?
6. В чому полягає ідея параметризації процедур ?
7. Чи бувають процедури: а) без механізму завершення обчислення , б) без повернення результату, в) без параметрів ?
8. В чому полягає різниця між процедурами та алгоритмами? Проілюструвати на прикладах цю різницю**.**
9. Які обставини послужили поштовхом до формалізації поняття алгоритму ?
10. Що таке масова проблема ?
11. Наведіть приклади масових проблем: а) розв'язних, б) нерозв'язних, в) відкритих.
12. Які основні вимоги до опису алгоритмів ?
13. Скінченність (фінітність) алгоритму – це: а) скінченність часу роботи алгоритму, б) скінченність опису алгоритму, скінченність функції керування алгоритму ?
14. Сформулюйте 10-ту проблему Гільберта.

Розділ 1. Процедури та їхнє математичне уточнення.

Сучасна теорія алгоритмів ще не сформувала загального формального визначення алгоритму. Причина такої ситуації – на поверхні. Моделі алгоритмів нерозривно пов'язані з конкретними мовами їхнього подання, а скільки моделей – стільки й мов.

Існуюча практика фіксувати якусь конкретну модель (штибу машин Тюрінга чи алгоритмів Маркова) і пов'язувати з нею загальне поняття алгоритму не вирішує головної проблеми – родового об'єднання моделей алгоритмів як класу певних об'єктів в межах спільної понятійної системи.

Доречно нагадати, що досить довго така ситуація існувала і в математиці з центральним її поняттям – функціональної залежності. Тільки з переходом математики на теоретико-множинну платформу і уточненні функціональної залежності як відповідності дозволило уніфікувати поняття функції і підійти до вивчення різних класів функцій з єдиних теоретико-множних позицій. Відомо як це вплинуло на розвиток математики у XX ст.

Щоб усунути першопричину роз'єднаності моделей алгоритмів, виникла ідея спочатку уточнити на теоретико-множинному рівні більш абстрактне (але близьке до алгоритмів!) поняття - процедури. Враховуючи, що воно не пов'язане з обмеженнями на способи подання, це повинно бути зробити простіше. А далі можна повернутися до алгоритмів і спробувати виокремити їхнє місце серед формальних процедур.

Як вже зазначалося, обчислення за процедурами і алгоритмами розгортаються в часі і супроводжуються виконанням певних дій. Цей час T – дискретний, тому його моменти можна ототожнити з натуральними числами *0,1,2…* і говорити про нульовий крок обчислення, перший, другий, тощо. Позначимо T_n початковий часовий відрізок з моментами часу від *0* до *n* включно. Нехай S та S_0 - сукупності станів та вхідних станів. Трійка $\Pi = (T, S, S_0)$ називається *обчислювальним простором*.

Кожному обчислювальному простору відповідає сукупність скінченних та нескінченних *процесів* - всюди визначених відображень вигляду $p: T_n \to S$, та $p: T \to S$. Процеси p будемо подавати

послідовностями станів $s_0, s_1, ...s_n$ та $s_0, s_1, ...$ відповідно, де $s_i = p(i), i \geq 0$ і зображати словами $s_0 s_1 ... s_n$ та $s_0 s_1 ...$ Нехай S^*, S^+ та S^∞ - сукупності відповідно усіх скінченних, непорожніх скінченних та нескінченних процесів. Початкові підпослідовності процесів будемо називати їхніми *префіксами.* У кожному нескінченному процесі для будь-якого $n > 0$ можна виділити його префікс довжини n і відповідну нескінченну хвостову частину.

Ми хочемо серед усіх процесів виділити *обчислювальні* процеси. В таких процесах поява чергового стану не випадкова, а певним чином залежить від поточного префіксу процесу і є результатом певної елементарної дії над його крайнім станом. Такі елементарні дії будемо розглядати як часткові операції на станах. (Випадок, коли дії не унарні і потребують не тільки крайніх станів, буде розглянуто далі в процедурах-численнях.)

Зафіксуємо певну сукупність $\Delta = \{F_i : S \to S, i = 0,1,...\}$ елементарних операцій на станах і нехай $\sigma = \{f_i, i = 0,1,...\}$ - сукупність сигнатурних символів для цих операції. Позначимо 2^σ множину всіх скінченних підмножин сигнатури σ. Центральним елементом процедур є тотальна *оперативна функція* вигляду $\delta : S^+ \to 2^\sigma$, яка регулює порядок застосування операцій в обчисленнях. Вона пов'язує з кожним префіксом обчислення скінченну підмножину можливих елементарних операцій для продовження обчислення. Префікси представляють історію обчислення на певний момент часу і ця історія може впливати як на вибір чергової елементарної операції, так і на закінчення процесу обчислення. Як вже зазначалося порожнє значення оперативної функції (ОФ) зупиняє обчислення.

Для прикладу, якщо взяти шахи, то станами виступають позиції гри і король може здійснити рокіровку тільки при умові, що він до цього моменту ще не ходив, а одним з варіантів закінчення гри є повторення тричі в процесі гри якоїсь з позицій.

Обчислювальною процедурою у просторі Π із сукупністю елементарних операцій Δ і оперативною функцією δ називається трійка $P = (\Pi, \Delta, \delta)$.

Це поняття (в дещо іншій формі), як і сама процедурна платформа, були запропоновані в [2].

Кожна обчислювальна процедура породжує певну сукупність обчислювальних процесів у просторі П. Домовимося обчислювальні процеси називати просто *обчисленнями*. Обчислення $p:T_n \to S$ та $p:T \to S$ за процедурою P з початковим станом $s \in S$ визначається рекурентно: $p_0 = s$ і для всіх $t > 0$ $p_t = F_t(p_{t-1})$, де F_t - одне зі значень оперативної функції δ на префіксі $p_0 ... p_{t-1}$.

Оперативна функція за попередніми станами обчислення $p_0 ... p_{t-1}$ визначає сукупність елементарних операцій, які можуть бути застосовані до стану p_{t-1} для отримання стану p_t. Будемо говорити, що обчислення p у момент часу t *переходить* від попереднього стану p_{t-1} до наступного стану p_t. Варіантів таких переходів може бути декілька, оскільки оперативна функція, як і самі елементарні операції, є багатозначною. Насправді, їх може бути: а) два і більше (*недетерміновані* процедури); б) рівно один (*детерміновані* процедури); в) жодного (оперативна функція для даного префіксу повертає порожнє значення чи елементарна операція на останньому його стані не визначені).

Зазначимо, що функція керування та оперативна функція в уточненні процедури тісно пов'язані. З кожним процесом обчислення $p_0, ..., p_i$ за процедурою пов'язана послідовність елементарних операцій з Δ $g_1, ..., g_i$, які виконуються на кожному кроці обчислення, розпочинаючи з першого кроку так, що $p_i = g_i p_{i-1}$. Тоді функцію керування можна можна було б покласти як $\bar{\delta}(s_0, g_1 ... g_{i-1},) = \delta(p_0 ... p_{i-1})$ і замінити нею оперативну функцію.

Найпростішими видом процедур є *темпоральні*. В них значення оперативної функції $\delta(p_0 ... p_{t-1})$ визначається просто довжиною t префіксу в обчисленні, самі ж проміжні стани при цьому не відіграють ніякої ролі. Тобто таку функцію δ можна визначити просто у вигляді $\delta : T \to 2^{\sigma}$.

Подібні оперативні функції характерні для різного роду службових інструкцій чи тих же уроків танців, де спочатку пропонують виконати першу фігуру, за нею – другу, потім - третю, тощо, а потім пропонують або зупинитися, або повторити все спочатку. Танці є також прикладом, так званих, *періодичних* обчислень, які відповідають темпоральним процедурам

з періодичною оперативною функцією, тобто такою, що $\delta(t)=\delta(t+n)$ для будь-якого моменту часу t і деякого фіксованого періоду обчислень $n\geq 1$.

Серед інших процедур окремо виділимо *автоматні*. Процедуру P назвемо *автоматною*, якщо значення її оперативної функції залежить тільки від останнього стану обчислення, тобто $\delta(p_0p_1...p_{t-1})=\delta(p_{t-1})$. Це найпоширеніший вид процедур. Більшість класичних алгоритмічних систем є саме автоматними.

Тепер щодо завершення обчислень і визначення їхніх результатів. Як вже зазначалося у Вступі, основний механізм завершення обчислень використовує порожню підмножину $\varnothing$ у значенні оперативної функції. Тобто, коли в процесі обчислення оперативна функція на поточному префіксі повертає значення $\varnothing$, обчислення припиняється. Воно також припиняється, коли елементарна операція на крайньому стані префікса не визначена. В обох випадках останній стан префіксу проголошується результатом обчислення.

Інший механізм зупинки обчислення пов'язаний з виділенням апріорі серед усіх станів процедури підмножини *вихідних* $F\subseteq S$ станів. Будемо визначати обчислювальні процедури з вихідними станами як четвірки $P=(\Pi,\Delta,\delta,F)$ і називати їх *ініціальними.*

В ініціальних процедурах обчислення закінчується, коли воно перший раз попадає у вихідний стан, який, як і в попередньому випадку, проголошується *результатом* обчислення. В обох випадках результат обчислень на вході s визначається $P^*(s)$. Він є єдиним у випадку детермінованих процедур.

Про відповідності $P^*:S_0\to S$ та $P^*:S_0\to F$ говорять, що їх *обчислює* процедура P, а саму процедуру називають *процедурою-функцією*. Зауважимо, що другий механізм завжди можна звести до першого, якщо 1) заключні стани назвати передзаключними, лишити їх статусу заключних і ввести єдиний новий заключний стан $q^*\notin S$, 2) до числа елементарних операцій долучити операцію f^* таку, що $f^*\in\delta(q), f^*(q)=q^*, q\in F$ і $\delta(q^*)=\varnothing$ і 3) результатом обчислення покласти не заключний його стан, а передзаключний.

Для темпоральних процедур діють ті ж механізми завершення обчислень і визначення результатів, що і в загальному випадку. Але вони можуть закінчуватись (наприклад, примусово) і по досягненню певного заданого моменту часу, вичерпанні часового ліміту на обчислення, тощо. Останнє має місце, коли оперативна функція δ визначена тільки на префіксах обмеженої довжини.

Якщо в повному обчисленні з вихідними станами зняти умову, щоб усі проміжні стани були не вихідними, то таке обчислення називається *гіперобчисленням*. Тоді результат такого процесу $P^{**}(s)$ складають усі його вихідні стани, що належать певному обчисленню з початком в s. Гіперобчислення характерні для числень.

Часто цікавлять не самі стани-результати, а тільки ті чи інші їхні складові[4] і тоді розглядають не саму відповідність, яку обчислює процедура, а її проекцію на ці складові.

Нехай B - довільна сукупність елементів, які назвемо *складовими* станів процедури. Нехай відображення вигляду $pr: S \to B$ є взаємно однозначним на початкових станах з S_0. Покладемо $B_0 = pr(S_0)$ та $B_1 = pr(F)$. Про відповідності вигляду $R^*: B_0 \to B_1$ та $R^{**}: B_0 \to B_1$, такі що $R^*(b) = pr(P^*(pr^{-1}(b)))$ та $R^{**}(b) = pr(P^{**}(pr^{-1}(b)))$, будемо говорити як про *проекції функцій*, які обчислює процедура P.

Типовим прикладом проекцій обчислювальних функцій є взяття певної компоненти в стані-кортежі, як це має місце в різних класах автоматів з вхідною стрічкою, зокрема і в машинах Тюрінга. В них стани становлять пари (внутрішній стан, слово на вхідній стрічці з виділеною доступною коміркою) і проекція на стані повертає тільки слово на стрічці.

Процедури P та Q називаються *еквівалентними* $(P \cong Q)$, якщо відповідності P^* та Q^*, які вони обчислюють, збігаються. Стан процедури назвемо *допустимим*, якщо він зустрічається серед станів деякого її результативного обчислення. Для відповідностей, які обчислює процедура, стани за межами підмножини її допустимих станів S_{acc} не важливі. Процедура, усі стани якої допустимі, називається *приведеною.*

[4] На практиці конкретні стани як конструктивні об’єкти мають певну структуру, елементами якої і є складові.

Багато прикладів обчислювальних процедур та алгоритмів можна знайти в шкільній програмі з алгебри та геометрії. Так, задачі пошуку розв'язку лінійних рівнянь та їх систем, розв'язок квадратних рівнянь, задачі пошуку за допомогою циркуля й лінійки середини відрізка, найбільшої спільної міри двох відрізків та найбільшого спільного дільника (НСД) розв'язуються за допомогою обчислювальних процедур [7]. Ці процедури, за винятком останніх, є темпоральними. Алгоритм Евкліда для пошуку найбільшої спільної міри двох відрізків та НСД - автоматний. Як вже зазначалося раніше, усі наведені процедури перестають бути алгоритмами, як тільки ми починаємо їх розглядати в контексті всіх дійсних чисел.

Є такі процеси обчислень, коли для отримання наступного стану процесу є необхідність звернутися не тільки до поточного стану, а й до деяких попередніх станів чи до аксіом. Це стосується, не оперативної функції, а елементарних операцій. Подібна ситуація зустрічається в деяких численнях. Дана обставина спонукає нас дещо понизити рівень абстракції і перейти від унарних елементарних операцій до $k-$*арних,* $k \geq 1$, вигляду $F^k : S \times ... \times S \to S$, які будемо називати ще *правилами виведення*.

Нехай $\Pi = (T, S, S_0)$ - довільний обчислювальний простір. Вхідні стани тепер будемо називатися – *аксіомами*. Замінимо унарні операції F_i в множині Δ на k_i-арні $F_i^{k_i}$, а сигнатуру σ- на сигнатуру $n-$арних функціональних символів $\{f_i^{k_i}, i = 0,1,...\}$. Оперативну функцію покладемо $\delta : S^* \to 2^\sigma$, тобто визначимо її і на порожньому префіксі. Нехай $F \subseteq S$ - довільна підмножиною вихідних станів. Четвірку $P = (\Pi, \Delta, \delta, F)$ будемо називати *процедурою-численням* або просто *численням* у просторі Π.

Обчислення в численнях є гіперобчисленнями і називаються *виводами* та пов'язується не з початковим вхідним станом, а з усіма аксіомами і з порожнім початковим префіксом ε . Обчислення $p : T \to S$ за процедурою-численням P визначається як послідовність станів $p_0, p_1, ...$ така, що для кожного $t \geq 0$ стан p_t будується за наступною схемою. Вибирається довільне правило виведення $f_i^{k_i}$ - з числа значень оперативної функції δ на префіксі $p_0 p_1 ... p_{t-1}$, тобто $f_i^{k_i} = \delta(p_0 p_1 ... p_{t-1})$, $f_i^{k_i} = \delta(\varepsilon)$ для нульового кроку. З попередніх станів $p_0, p_1, ..., p_{t-1}$ та аксіом з S_0 формується довільний кортеж

$(q_1,...,q_{k_i})$ з області визначення правила $f_i^{k_i}$ і тоді $p_t = F_i^{k_i}(q_1,...,q_{k_i})$. Таким чином, може існувати декілька варіантів наступного стану p_t. Для кожного вибраного кортежу – свій.

Виведення є гіперобчисленнями, про останній його стан, якщо він є заключним, говорять, що він *виводиться з посилок*, а саме виведення називають його *доведенням з посилок.* Але до результатів виведення (як гіперобчислення) долучаються і всі його проміжні вихідні стани. Гіперобчислення може зупинятися і з причини невизначеності оперативної функції на останньому стані. Заключні стани, які виводяться з аксіом називаються *теоремами*.

В численнях на передній план виступає сукупність усіх теорем $P^{**}(\varepsilon)$, які виводяться з порожнього префікса і аксіом. Ця сукупність називається *формальною теорією* числення і позначається $Th(P)$. Говорять, що процедура-числення *породжує* свою теорію. Найпростішим класом процедур-числень є автоматні числення з унарними елементарними правилами виведення, до яких належать різні класи формальних граматик, системи Поста, тощо, про які буде йти мова в Розд.4.

Як простий, але важливий приклад числень розглянемо *канонічні числення*, пов'язані з алгебрами. Вони дозволяють формально побудувати кожен з елементів алгебри з породжувальних її елементів. Нехай $A = (A; f_1^{k_1},..., f_m^{k_m},...)$ [5] довільна алгебра з множиною породжувальних елементів $A_0 \subset A$ сигнатури σ. Її канонічне числення будемо позначати $C(A)$. Станами числення $C(A)$ виступають елементи алгебри, аксіомами – породжувальні її елементи, правилами виведення – операції алгебри. Оперативна функція числення δ є константою – на будь-якому префіксі обчислення повертає всю сукупність операцій алгебри.

Теорія канонічного числення алгебри $Th(C(A))$ збігається з усім носієм алгебри A. Це випливає з того, що аксіоми складають систему породжувальних елементів алгебри. Таким чином, кожний елемент алгебри має доведення в численні $C(A)$ і воно надає шлях його побудови.

У багатьох численнях не виділяють вихідні стани і до теорем відносять, як у канонічних численнях алгебр, усі виведені стани. Проте це

[5] A позначає як саму АС, так і її носій, але з контексту завжди буде зрозуміло, що мається на увазі.

не завжди можливо чи доцільно. Введення заключних станів в теоріях дозволяє відокремити теореми від решти (проміжних) станів.

Процедури, які обчислюють функції, разом з процедурами-численнями, коли це не принципово, будемо називати загальним терміном *процедура*.

Таким чином, ми побудували загальну математичну модель поняття процедури. Обчислювальні процедури є спеціальним підкласом динамічних дискретних систем [8]. Тепер, щоб отримати математичне визначення алгоритму, залишилося формально виокремити алгоритми серед обчислювальних процедур.

Контрольні запитання

1. Що таке обчислювальний простір ?
2. Що таке процес в обчислювальному просторі і його префікси?
3. У чому полягає різниця між процесом у просторі і обчислювальним процесом?
4. Що таке елементарна дія?
5. Що таке процедура?
6. Що таке обчислення за процедурою ?
7. Що таке оперативна функція і яка її роль в обчисленнях?
8. Що таке детерміновані і недетерміновані процедури?
9. Що таке темпоральні та періодичні автоматні процедури? Навести приклади.
10. Що таке автоматні процедури? Навести приклади.
11. Що таке ініціальна процедура?
12. Що таке гіперобчислення ?
13. Що таке проекції функцій, які обчислює процедура ?
14. Які процедури називаються еквівалентними ?
15. Що таке процедура-числення ?
16. Що таке виведення в численні ?
17. Що таке аксіоми і теореми в численнях ?
18. Що таке канонічні числення в алгебрах ?

Розділ 2. Алгоритми як конструктивні процедури.

Як ми вже наголошували в Розд. 1**,** єдина відмінність між процедурами й алгоритмами полягає у їхньому описі – опис алгоритмів має бути скінченним і повним. Об'єкти, які мають такий опис, отримали назву *конструктивних.*

Поняття конструктивного об'єкту належить до первинних природничо-наукових понять [9,10], тому поняття алгоритму є похідним від нього.[6] Це означає, що 1) формальному уточненню поняття алгоритму має передувати вибір тієї чи іншої конкретної моделі конструктивності (для станів, елементарних операцій, оперативної функції та характеристичних функцій підмножин вхідних та заключних станів) і 2) поняття алгоритму завжди буде не абсолютним, а відносним – залежати від вибраної у п.1) моделі конструктивності.

Сказане яскраво ілюструє палітра конструктивних об'єктів в традиційних моделях алгоритмів. Всі вони базуються на таких *apriori* вибраних конкретних конструктивних об'єктах як числа, послідовності, слова, скінченні графи тощо. Основу кожного формального класу конструктивних об'єктів складає певна фінітна дескриптивна знакова система (ДС)[11].

Фінітність ДС передбачає обов'язкову скінченність її термів[7]. У фінітній системі кожний терм може бути побудований за скінченну кількість кроків за правилами ДС з первинних символів, які складають алфавіт системи, а повнота опису функції в ДС має гарантувати можливість реального обчислення її значень за її термом (у разі, якщо значення існує).

Таке обчислення має відбуватися за правилами, які є невід'ємною складовою дескриптивної системи. Це означає що сама ДС потребує наявності в ній певної моделі алгоритму для обчислення представлених в ній функцій. Але тут не виникає «зачарованого кола», тому що мова йде тільки про конкретний частковий випадок алгоритмічності функцій, який задається

[6] Існує й дуальний підхід, коли первинним поняттям є алгоритм, а похідним – конструктивний об'єкт, як такий, для якого існує алгоритм його побудови.

[7] Терми – це синтаксичні конструкції ДС, які подають об'єкти ДС. Зазначимо, що ДС не обов'язкову має бути словарною. Термами, наприклад, можуть, бути скінченні графи, тощо.

незалежно від загального поняття. Як ми побачимо в Розд.4, часто для опису, наприклад, оперативних функцій достатньо й скінченних таблиць. Якщо всі елементи процедури – стани, елементарні операції, оперативна та характеристичні функції початкових та заключних станів є конструктивними об'єктами в рамках даної дескриптивної системи, то процедура називається *конструктивною в ДС* або просто - *конструктивною*, якщо така ДС просто існує.

Таким чином, ми прийшли до наступного формального визначення процедурного алгоритму.

Процедурний алгоритм - це конструктивна процедура.

Найбільшою проблемою при переході від процедур до алгоритмів є фінітний опис оперативної функції, яка за означенням, є семантично нескінченним об'єктом.

Якраз саме з цією проблемою пов'язана головна технологічна складність побудови програм для ЕОМ.

Спробуємо конкретизувати дану проблему і нагадаємо, що два елементи з області визначення довільної функції *ядерно еквівалентні,* якщо значень функції на цих елементах збігаються, а *індексом* відношення еквівалентності називається потужність множини її фактор-класів.

Нас цікавить ядерна еквівалентність оперативних функцій конструктивних процедур. Дана еквівалентність для кожної такої процедури завжди має скінченний індекс. Це випливає зі скінченності області значень оперативної функції і означає, що таку функцію можна задати скінченною *оперативною таблицею* (ОТ), в першому стовпчику якої перераховані всі фактор-класи ядерної еквівалентності, у другому – значення оперативної функції на префіксах обчислень відповідного фактор-класу.

Щоб знайти значення оперативної функції на певному префіксі обчислення достатньо знайти рядок таблиці, до якого він належить і взяти одну з операцій з другого стовпчика. Проблема в тому, що скінченність представлення оперативної таблиці ще не означає її фактичної скінченності, тому що класи еквівалентних станів є нескінченними.

Оперативну таблицю назвемо *розв'язною*, якщо існує алгоритм, який дозволяє знайти в ній рядок, до якого належить довільний префікс або дати відповідь, що такого рядка в таблиці немає. Розв'язність оперативної таблиці гарантує існування скінченного опису оперативної функції.

Як зазначалося вище, це разом з конструктивністю станів і елементарних операцій робить процедуру алгоритмом. Вірно і обернене твердження, що у конструктивної ОФ класи ядерної еквівалентності – розв'язні.

Отже, ми готові конкретизувати наше формальне визначення алгоритму.

Алгоритм – це конструктивна процедура з розв'язною оперативною таблицею.

Як вже зазначалося, таке визначення алгоритму не містить «зачарованого кола», тому що кожного разу мова йде не про загальну розв'язність множин та конструктивність операцій, а тільки про окремі їхні часткові випадки, кожен з яких в разі потреби може бути математично визначеним без апеляції до загального поняття.

Виділимо один важливий випадок розв'язності оперативних таблиць. Нагадаємо, що кожний стан конструктивної процедури має певне скінченне синтаксичне подання як конструктивний об'єкт, тому і префікси як послідовності станів також мають своє синтаксичне подання (у вигляді відповідної послідовності позначень станів). На практиці, ядерно еквівалентні префікси часто мають свою особливу синтаксичну ознаку, при цьому у різних класів ці ознаки різні.

Нехай в оперативній таблиці m рядків і $\{\omega_1,...,\omega_m\}$ - сукупність всіх вказаних ознак. Поставимо оперативній таблиці у відповідність цю ж таблицю з заміною фактор-класів на їхні ознаки ω_i і назвемо її *синтаксичною* оперативною таблицею (СОТ). Тепер пошук значення оперативної функції буде зводитися до з'ясування синтаксичної ознаки префіксу і взяття значення у другій комірці відповідного рядка в СОТ. Як ми побачимо далі, для всіх традиційних алгоритмічних систем такі синтаксичні ознаки існують і в більшості випадків вони достатньо прості.

Будемо називати далі алгоритми як конструктивні процедури А-*процедурами* – скорочення від алгоритмічних процедур.

Як процедури, А-процедури можуть бути детермінованими й недетермінованими. Функції (відповідності), які обчислюють А-процедури утворюють клас А-*процедурно обчислювальних* функцій. Теорії, які породжуються А-численнями, утворюють клас А- *перелічних* множин.

Наведемо основні властивості А-процедур, які безпосередньо випливають з їхнього формального визначення.

Масовість. А -процедура може бути застосована до будь-якого вхідного стану обчислювального простору.

Дискретність. Обчислення за А-процедурою відбуваються в дискретному часовому просторі.

Елементарність. В кожний момент часу в обчисленні виконується одна елементарна операція з фіксованої сукупності таких операцій.

Визначеність. Порядок застосування елементарних операцій в обчисленні не довільний, а визначається операційною функцією за поточним префіксом обчислення.

Результативність. У кожній *А*-процедурі є механізм завершення обчислень.

Фінітність. Скінченність подання станів, елементарних операцій та оперативної функції.

Відносність. Визначення А-процедури відносне. Це пов’язано не тільки традиційно з тим, що у деяких випадках окремі чи всі елементарні операції можуть виступати як параметри з наступною їх актуалізацією. Більше того, деякі з них можуть і не актуалізуватися перед застосуванням процедури. Тоді вони називаються *оракулами*, а відповідні А-процедури - А-процедурами *з оракулами* [6]. А й з його залежністю від типу конструктивності ОТ.

Якщо порівняти загальне поняття алгоритму і його процедурний аналог, то побачимо багато спільного. Зазвичай, загальне поняття пов’язують зі способом подання сукупності обчислювальних процесів для розв’язку задач певного класу [1]. Характеристичні властивості цього

способу такі, як фінітність, масовість, дискретність, елементарність, результативність та керованість фактично притаманні й процедурним алгоритмам. Розходження стосуються тільки визначеності загальних алгоритмів і детермінованості - процедурних. Ці властивості стосуються вибору чергової дії в обчисленні.

У випадку А-процедур сукупність цих можливих дій визначає спеціальна оперативна функція, а в алгоритмах даний вибір поточної дії не уточнюється, а замість цього стверджується, що поточна дія чи можливі дії на кожному кроці обчислення є строго визначеними і детерміновано випливають з історії обчислення (*керованість* діями).

До цієї історії входить початковий стан і всі попередні дії обчислення. В процедурах історія обчислень представлена послідовністю його станів, така послідовність дозволяє за допомогою оперативної функції побудувати і історії відповідних дій, а значить і забезпечити керованість діями. Тому процедурні алгоритми є підкласом загальних алгоритмів.

Але керованість дій в алгоритмі дає підстави стверджувати існування, нехай і неявної його функції керування $\bar{\delta} : S_0 \times \Delta^* \to \Delta$, якщо для $u \in \Delta^*$ покласти значенням $\bar{\delta}(s_0, u)$ сукупність тих строго визначених за алгоритмом дій на історії $s_0, u \in \Delta^+$.

Тобто, щоб знайти значення функції керування алгоритму $\bar{\delta}(s_0, u)$ достатньо здійснити з початкового стану s_0 за алгоритмічними правилами скінченну кількість кроків обчислення згідно історії u і взяти ті елементарні операції, які визначені алгоритмом на останньому кроці для застосування до поточного стану. Дані кроки фінітно описані в алгоритмі, що забезпечує і фінітність опису функції $\bar{\delta}$, а значить і її конструктивність.

Зі сказаного випливає, що кожний загальний алгоритм має свій процедурний еквівалент, тобто алгоритмічну процедуру, яка породжує ту саму сукупність обчислень, що й даний алгоритм.

Отже, ми можемо стверджувати, що має місце наступна

Лема 2.1. Для кожного інтуїтивного алгоритму існує еквівалентний процедурний алгоритм.

Якщо назвати функції, які обчислюються алгоритмами *ефективними,* то як випливає з Леми класи ефективних функцій і функцій, які обчислюються процедурними алгоритмами, збігаються.

Контрольні запитання

1. Що таке конструктивний об’єкт та конструктивна функція?
2. Що таке ядерна еквівалентність оперативної функції ?
3. Що таке оперативна таблиця процедури ?
4. Що таке розв’язність оперативної таблиці ?
5. Що таке алгоритмічна процедура ?
6. Про яке «зачароване коло» йдеться при визначенні алгоритму як процедури ?
7. Що таке синтаксична оперативна таблиця ?
8. Перерахувати основні властивості А-процедур.
9. Чим обумовлена відносність А-процедур ?

Розділ 3. Загальні властивості процедур та алгоритмів.

Позначимо ПАФ клас функцій, обчислюваних процедурними алгоритмами. Розглянемо деякі загальні властивості процедур, А-процедур та обчислюваних ними функцій. Характеристичну функцію χ_0 множини вхідних станів процедури назвемо *вхідною*. Як зазначалося, можна обмежитися процедурами та алгоритмами без заключних станів. Нехай $Fin = \{s : \delta(us) = \varnothing\}$ і χ_{Fin}- вихідна характеристична функція множини Fin. Нагадаємо, всі три функції оперативна, вхідна та вихідна – тотальні.

Нехай $\varphi : S_1 \to S_2$- певне відображення станів. Воно переноситься і на відображення префіксів з цих станів як $\varphi(s_1...s_n) = \varphi s_1..., \varphi s_n$. Бієкція $\varphi : S_1 \to S_2$ - між множинами станів процедур $P_1 = (\Pi_1, \Delta_1, \delta_1)$ та $P_2 = (\Pi_2, \Delta_2, \delta_2)$, де $\sigma_1 = \{f_1,..., f_n\}, \sigma_2 = \{g_1,..., g_m\}$- сигнатури цих процедур, називається *ізоморфізмом процедур* (а самі процедури - *ізоморфними),* якщо елементарні операції та оперативні функції процедур діють узгоджено, тобто для кожного префікса $ps \in S_1^* S_1$ і для кожного сигнатурного символу $f \in \delta_1(ps)$ існує такий сигнатурний символ $g \in \delta_2(\varphi ps)$, що $\varphi F(s) = G(\varphi s),$ де $F \in \Delta_1, G \in \Delta_2$ - елементарні операції, які відповідають сигнатурним символам $f \in \sigma_1, g \in \sigma_2$ і навпаки, якщо f і g поміняти місцями і взяти обернене відображення $\varphi^{-1} : S_2 \to S_1$.

Вочевидь, функції, обчислювані ізоморфними процедурами, теж узгоджені, тобто $\varphi P_1^*(s) = P_2^*(\varphi s)$. Ізоморфні процедури будемо називати *варіантами* одна одної, якщо $S_1 \cap S_2 = \varnothing$.

Якщо у визначенні ізоморфізму процедур бієкцію $\varphi : S_1 \to S_2$ замінити на сур'єкцію, то отримає поняття гомоморфізму А-процедур. Візьмемо фактор-процедуру $[P_1] = ([\Pi_1]_\varphi, [\Delta_1]_\varphi, [\delta_1]_\varphi)$, побудовану на фактор-класах ядерної еквівалентності гомоморфізму φ так, що $[F_i]([s]) = [F_i(s)]$ і $[\delta_1]([s]) = \delta_1(s)$. Тоді, за побудовою, бієкція $[\varphi] : [S_1] \to S_2$, $[\varphi]([s]) = \varphi s$, буде ізоморфізмом процедури $[P_1]$ на процедуру P_2.

Серед процедур особливе місце займають, так звані, вільні процедури. Візьмемо довільну процедуру $P = (\Pi, \Delta, \delta)$ сигнатури σ і зафіксуємо певну систему породжувальних елементів $X \subseteq S$ для сукупності її станів відносно

операцій з Δ та сигнатуру констант $\sigma^0 = \{c_1,...,c_n,...\}$ для породжувальних елементів. *Термами* сигнатури σ назвемо слова мови $\mathrm{T}_\sigma = \sigma^*\sigma^0$. Кожний стан $s \in S$ представляється одним з термів $f_{i_1}...f_{i_n}c$ таких, що $F_{i_1}(...(F_{i_n}x)...) = s$ і c - константа для x. Визначимо наступну сукупність елементарних операцій $\bar{\Delta} = \{\bar{f}_1,..., \bar{f}_n,...\}$ на термах сигнатури $\bar{\sigma} = \sigma \cup \sigma^0$: $\bar{f}_i(f_{i_1}...f_{i_n}c) = f_i f_{i_1}...f_{i_n}c$. Обчислювальний простір зі станами-термами сигнатури $\bar{\sigma}$ будемо називати *вільним.* Процедуру у вільному просторі з елементарними операціями на термах будемо називати *вільною*.

Теорема 3.1 (про гомоморфізм процедур).

Кожна процедура $P = (\Pi, \Delta, \delta)$ сигнатури $\bar{\sigma}$ є гомоморфним образом деякої вільної процедури сигнатури $\bar{\sigma}$.

Доведення. Візьмемо вільну процедуру $\bar{P} = (\bar{\Pi}, \bar{\Delta}, \bar{\delta})$, довільну бієкцію $\varphi: \sigma_0 \to X$ і поширимо її на всю сукупність термів: $\varphi(f_{i_1}...f_{i_n}c) = F_{i_1}(...(F_{i_n}(\varphi c)...)$. Покладемо $\bar{\delta}(t) = \delta(\varphi t)$ для кожного префіксу $t \in \mathrm{T}^*_{\bar{\sigma}}$. За побудовою, $\varphi: \mathrm{T}^*_{\bar{\sigma}} \to S$ є гомоморфізмом вільної процедури $\bar{P}$ на процедуру A.◄

Наслідок 3.1. Кожна процедура $P = (\Pi, \Delta, \delta)$ сигнатури $\bar{\sigma}$ ізоморфна певній фактор-процедурі вільної процедури сигнатури $\bar{\sigma}$. ◄

Визначимо на множині функцій і предикатів на станах операції системи алгоритмічних алгебр Глушкова (САА): множення $\circ$, розгалуження $(\to|)$ та ітерації $\{\to\}$[13]. Нехай $f, g: S \to S, p: S \to \{0,1\}$ довільні функції і предикат на станах. Тоді, за означенням, $f \circ g(a) = f(g(a))$, $(p \to f \mid g)(a) = f(a)$, якщо $p(a) = 1$ і $(p \to f \mid g)(a) = g(a)$, якщо $p(a) = 0$ і $\{p \to f\}(a) = f^n(a)$, якщо $p(f^n(a)) = 1$ і $p(f^i(a)) = 0$ для всіх $i = 0,1,..,n-1$. Функції і предикати, отримані за допомогою цих операцій називаються *регулярними*.

Тотальну функцію вигляду $\alpha: A \to [1..n]$ назвемо *перемикачем*, а $(n+1)$–арною операцією детермінованого вибору з перемикачем α - операцію *case* таку, що $case(\alpha, g_1,..., g_n) = g_i$, якщо $\alpha(a) = i$. Аналогічно визначається операція недетермінованого вибору з перемикачем $\alpha: A \to 2^{[1..n]}$ на підмножинах функцій $\Delta_1,...,\Delta_n$: $case(\alpha, \Delta_1,..., \Delta_n) = g_j$, якщо $\alpha(a) = i$ і $g_j \in \Delta_i$.

САА (регулярні функції), отримані з залученням детермінованого і недетермінованого виборів називаються *розширеними (розширеними регулярними).*

Теорема 3.2 (про регулярізацію).

Нехай $P=(\Pi,\Delta,\delta)$ автоматна процедура з множиною елементарних операцій $\Delta=\{F_1,...,F_n,...\}$. Функція P^* є розширеною регулярною функцією відносно елементарних операцій, оперативної функції δ та вхідної і вихідної функцій χ_0, χ_{Fin}.

Доведення. В поданні функції δ може фігурувати тільки скінченна кількість елементарних функцій. Занумеруємо $\Delta_0=\varnothing,\Delta_1,\Delta_2,...,\Delta_k$ всі варіанти підмножин-значень оперативної функції δ і будемо вважати, що оперативна функція повертає не підмножини, а їхні номери. Тоді за означенням, $P^*=(\chi_0\to\{\chi_{Fin}\to case(\delta,\Delta_0,...,\Delta_k)\})$.◄

Теорема про регуляризацію має місце і для неавтоматних процедур.

Наслідок 3.2. Для неавтоматних процедур теж має місце теорема про регуляризацію. Нехай $P=(\Pi,\Delta,\delta)$ - довільна процедура. Покладемо $\tilde{\Pi}=(T,S^*,S_0)$ $\tilde{\Delta}=\{\tilde{F}_1,...,\tilde{F}_n,...\}$, де $\tilde{F}_i:S^*\to S^*$ і $\tilde{F}_i(s_1...s_i)=s_1...s_iF_i(s_i)$, $last(s_1...s_i)=s_i$. Розглянемо автоматну процедуру $\tilde{P}=(\tilde{\Pi},\tilde{\Delta},\delta)$. Тоді за побудовою і теоремою про регуляризацію автоматних процедур маємо $P^*=\tilde{P}^*\circ last=$ $(\chi_0\to\{\chi_{Fin}\to case(\delta,\tilde{\Delta}_0,...,\tilde{\Delta}_k)\}\circ last$.◄

Для аналізу й синтезу формальних алгоритмічних систем використовується алгебра Чорча та її підалгебри [1,17]. Нехай $F^n:N^n\to N$ клас всіх n-арних натуральних функцій, $F(N)=\bigcup_{n=0}^{\infty}F^n$ - клас всіх натуральних функцій.

Операціями алгебри Чорча є операції суперпозиції $S^{n+1},n\ge1$, примітивної рекурсії R та мінімізації M. Операція S^{n+1} n-арній функції $g(x_1,...,x_n)$ та n функціям $g_1(x_1,...,x_m)$ $g_n(x_1,...,x_m)$ одної і тої ж арності зіставляє функцію $f(x_1,...,x_m)$ $=g(g_1(x_1,...,x_m),...,g_n(x_1,...,x_m))$, яку будемо позначати $S^{n+1}(g,g_1,...,g_n)$.

Операція *примітивної рекурсії* R n-арній функції g та $(n+2)$-арній функції h зіставляє $(n+1)$-арну функцію f, яку позначають $R(g,h)$, що задається співвідношеннями:

$$f(x_1,...,x_n,0) = g(x_1,...,x_n)$$

$$f(x_1,...,x_n,y+1) = h(x_1,...,x_n,y,f(x_1,...,x_n,y))$$

Це означає, що для обчислення значення $f(a_1,...,a_n,b)$ будується рекурентна послідовність чисел $c_0,c_1,...,c_b$ така, що $c_b = f(a_1,...,a_n,b)$ і $c_0 = g(a_1,...,a_n)$, $c_i = h(a_1,...,a_n i-1,c_{i-1})$, $i = 1,...,n$.

Операція мінімізації M кожній $(n+1)$-арній функції g зіставляє n-арну функцію f, яку позначають $M(g)$ і задається співвідношенням $f(x_1,...,x_n) = \mu_y(g(x_1,...,x_n) = 0)$, де права частина визначає найменший розв'язок рівняння $g(x_1,...,x_n,y) = 0$ відносно y при умові, що значення функції $g(x_1,...,x_n,i)$ для всіх $i < y$ визначені.

Алгеброю Чорча називається підалгебра алгебри натуральних функцій $(F(N); S^{n+1}, n = 1,2,..., R^2, M^1)$, породжена *базовими функціями* $0(x) = 0, s(x) = x+1$ та функціями-селекторами $I_m^n(x_1,...,x_n) = x_m$, де $n \geq m \geq 1$.

Функції алгебри Чорча називають *частково рекурсивними* (ЧРФ). Виділяють також підкласи *примітивно рекурсивних* функцій (ПРФ) – функцій, для породження яких використовуються тільки операції суперпозиції та примітивної рекурсії - і підклас *рекурсивних* функцій (РФ) – тотальних ЧРФ.

Множини натуральних чисел називаються *примітивно рекурсивними* та *рекурсивними*, якщо їхні характеристичні функції належить класу ПРФ та відплвідно РФ та - *частково рекурсивними*, якщо їхня часткова характеристична функція належить класу ЧРФ.

Алгебра Чорча має злічену сукупність часткових операцій, в яких аргументи взаємозалежні з точки зору їхньої арності. Це накладає відбиток на її канонічне алгебричне числення. Зокрема, оперативна функція δ числення вже не є константою, а повертає для даного префіксу обчислення скінченну підмножину операцій $\{S^{n+1}, n = 1,2,...m, R^2, M^1\}$, де m - найбільша арність функцій з префіксу обчислення.

Теорія канонічного числення алгебри Чорча збігається з класом ЧРФ всіх частково-рекурсивних функцій, а саме числення є численням.

Розглянемо канторову нумерацію $c^m(x_1,...,x_m)$ натуральних векторів декартового добутку N^m і функції-проекції $c_{m1}(x),...,c_{mm}(x)$ такі, що $c^m(c_{m1}(x),...,c_{mm}(x))=x$. Вона дозволяє кожній ЧРФ $f^m(x_1,...,x_m)$ поставити у відповідність її унарний аналог $\overline{f}(x)=f^m(c_{m1}(x),...,c_{mm}(x))$. За означенням, $f^m(x_1,...,x_m)=\overline{f}(c^m(x_1,...,x_m))$. Функції $c^m, c_{m1},...,c_{mm}$ є примітивно рекурсивними, тому унарні аналоги ЧРФ функцій є частково рекурсивними. Це дозволяє при розгляді частково рекурсивних функцій обмежитися тільки унарними функціями.

Далі під класом ЧРФ будемо розуміти підмножину унарних ЧРФ. На основі канторової нумерації векторів $c^m(x_1,...,x_m)$ будується канторова нумерація $\nu: N^* \to N$ довільних послідовностей натуральних чисел. Вона теж примітивно рекурсивна.

Процедури в натуральному часовому просторі називаються *натуральними.* Нехай $P=(\Pi,\Delta,\delta)$ - довільна натуральна процедура. Покладемо $\Omega_0=\Delta\cup\{\delta,\chi_0,\chi_{Fin}\}$ Нумерація ν дозволяє замінити префікси натуральних процедур і значення їхніх оперативних функцій числовими номерами і розглядати оперативні функції як просто унарні числові $\overline{\delta}(x)=\delta(\nu^{-1}(x))$.

Якщо долучити до базових ЧРФ підмножину Ω довільних числових функцій, то отримаємо поняття *частково рекурсивної функції відносно множини* Ω і клас функцій Ω- ЧРФ.

Теорема 3.3. Клас функцій, обчислюваних натуральними процедурами збігається з класом Ω_0-ЧРФ.

Доведення. Клас Ω_0-ЧРФ замкнений відносно операцій розширеної СААП (див. Вправу 3.7). Тоді з Теореми 3.2 про регуляризацію випливає, що функції, обчислювані натуральними процедурами, належать класу Ω_0-ЧРФ.

Обернене включення випливає з того, що довільна Ω_0-ЧРФ обчислюється регістровою машиною з оракулами, які повертають значення функцій з Ω_0, а такі машини є підкласом натуральних процедур .(Див.

Розд.4) ◄

Нехай $P=(\Pi,\Delta,\delta)$ - довільна процедура. Конструктивність процедури має гарантувати існування ефективної нумерації станів $\beta: N\to S$ та рекурсивність натуральних еквівалентів оперативної функції та вхідної і вихідної функцій. Покладемо функцію кодування $\chi: S\to N$, яка за станом повертає його найменший номер в нумерації β і розглянемо функцію $P_N^*: N\to N$ таку, що $P_N^*(n)=\chi P^*(\beta n)$. Назвемо функцію P_N *натуральним еквівалентом* функції P^*. Операцію $()_N$, яка ставить функції $f: S\to S$ її натуральний еквівалент f_N, назвемо операцією *натуралізації*. Нехай $P=(\Pi,\Delta,\delta)$ - довільна процедура сигнатури σ. Назвемо процедуру $P_N=(\Pi_0,\Delta_N,(\delta)_N)$ сигнатури σ в натуральному просторі сигнатури Π_0 з множиною вхідних станів $\beta(S_0)$ та натуральними еквівалентами елементарних операцій та оперативної функції *натуральним еквівалентом* P. За побудовою, натуральний еквівалент процедури P обчислює функцію, яка є натуральним еквівалентом функції P^*.

Теорема 3.4. Натуральні еквіваленти функцій, обчислювальних процедурами, утворюють клас $(\Omega_0)_N$-ЧРФ. Якщо всі функції з $(\Omega_0)_N$ є ЧРФ, тоді клас $(\Omega_0)_N$-ЧРФ збігається з класом ЧРФ.

Доведення. Випливає з Теореми 3.3. ◄

Прямою сумою $A\oplus B$ двох множин називається множина $\bar{A}\cup\bar{B}$, де $\bar{A},\bar{B}$ - множини A,B, елементи яких проіндексовані відповідно індексом 1 та 2, тобто $\bar{A}=\{a_1: a\in A\}, \bar{B}=\{b_2: b\in B\}$ За означенням, $\bar{A}\cap\bar{B}=\varnothing$,$(A\oplus B)_1=\bar{A}, (A\oplus B)_2=\bar{B}$. Нехай $^{-1}$ - операція *розіндексування* елементів прямої суми, тобто $a_1^{-1}=a, b_2^{-1}=b.$ Звертаємо увагу, що при індексації і розіндексації об'єктів їхні значення не змінюються – у першому випадку у елемента з'являється числовий індекс, у другому - він усувається. *Прямою сумою* $F\oplus G$ функцій $F: A\to C, G: B\to D$ називається функція $F\oplus G: A\oplus B\to C\oplus D$ така, що $F\oplus G(x)=(F(x^{-1}))_1,$ якщо $x\in\bar{A}$ і $F\oplus G(x)=(G(x^{-1}))_2,$ якщо $x\in\bar{B}$. Як бачимо, перед обчисленням значення прямої суми $F\oplus G$ відбувається розіндексація аргумента, а потім навпаки – відповідна індексація результату. Це дозволяє не змінювати самі функції F,G. Покладемо $\bar{F}(x)=(F(x^{-1}))_1, \bar{G}(x)=(G(x^{-1}))_2$. Тоді $F\oplus G(x)=\bar{F}(x),$ якщо $x\in\bar{A}$ і $F\oplus G(x)=\bar{G}(x),$ якщо $x\in\bar{B}$. Покладемо

$\bar{\Delta}_1 = \{\bar{F}_1, ..., \bar{F}_n, ...\}, \bar{\Delta}_2 = \{\bar{G}_1, ..., \bar{G}_n, ...\}$. *Прямою сумою* процедур $P = (\Pi_1, \Delta_1, \delta_1)$ та $Q = (\Pi_2, \Delta_2, \delta_2)$ в обчислювальних просторах $\Pi_1 = (T, S, Bx_1)$ та $\Pi_2 = (T, S, Bx_2)$ називається процедура $P \oplus Q = (\Pi, \bar{\Delta}_1 \cup \bar{\Delta}_2, \delta_1 \oplus \delta_2)$ в обчислювальному просторі $\Pi = (T, S \oplus S, Bx_1 \oplus Bx_2)$. Вона обчислює функцію $(P \oplus Q)^* : Bx_1 \oplus Bx_2 \to S_1 \oplus S_2$. Аналогічно, визначається пряма сума для будь-якого числа $n, n > 2$, функцій та процедур

Іноді виникає необхідність організувати послідовну взаємодію кількох процедур з їхньої прямої суми. Якщо, наприклад, взяти суму $P \oplus Q$ і його активна процедура перейде в заключний стан, то обчислення, за означенням, зупиниться.

Цього не станеться, якщо замість припинення роботи перейти в інший компонент і розпочати там нове обчислення на проіндексованому відповідним чином проміжному результаті. Таку можливість забезпечують інтерфейсні операції $\varphi_{12} : (S)_1 \to (S)_2, \varphi_{21} : (S)_2 \to (S)_1$ такі, що $\varphi_{12}(x) = ((x)^{-1})_2, \varphi_{21}(x) = ((x)^{-1})_1$ і долучення їх до можливих значень оперативних функцій (операцію φ_{12}- до значень функції δ_1, операцію φ_{21}- до значень функції δ_2). Після переходу в нову процедуру оперативна функція $\delta_1 \oplus \delta_2$ буде «ігнорувати» префікс попереднього обчислення і просто його продовжить «з нуля» в новій компоненті з того стану, який надасть інтерфейсна операція. Процедура завершить роботу, як і раніше, коли зупиниться одна з компонент.

Щоб реалізувати сказане, для всіх $w \in (S^*)_1 \cup (S^*)_2$, $r \in \bar{S}_1 \cup \bar{S}_2$ таких, що w і r мають різні індекси, покладемо: $\delta_1 \oplus \delta_2(wr) = \delta_1 \oplus \delta_2(r)$. У цьому випадку попередня історія обчислення в іншій компоненті не грає ролі.

Будемо називати таку нову процедуру з операціями-переходами *квазі-прямою* сумою $P \hat{\oplus} Q$ процедур.

Декартовим добутком функцій $F : A \to B, G : C \to D$ називається функція $F \times G : A \times C \to B \times D$ така, що $F \times G(a,c) = (F(a), G(c))$. Аналогічно визначається і декартовий добуток n функцій $F_1 \times ... \times F_n$. *Декартовим добутком* процедур $P = (\Pi_1, \Delta_1, \delta_1)$ та $Q = (\Pi_2, \Delta_2, \delta_2)$ в обчислювальних просторах $\Pi_1 = (T, S, Bx_1)$ та $\Pi_2 = (T, S, Bx_2)$ називається процедура називається процедура $P \times Q = (\Pi, \Delta_1 \times \Delta_2, \delta_1 \times \delta_2)$ в обчислювальному просторі $\Pi = (T, S \times S, Bx_1 \times Bx_2)$, яка

моделює паралельну роботу процедур-компонент. Вона зупиняється тільки, коли зупиняться обидві процедур. Опустимо формально деталі (див. Вправу 10). *Тотожньою* процедурою в обчислювальному просторі Π називається процедура $I=(\Pi,\{f_0\},\delta,S)$ з єдиною елементарною операцією – тотожньою $f_0(s)=s$.

Теорема 3.5 (про замкненість). Клас функцій, обчислювальних процедурами замкнений відносно операцій прямої та квазі-прямої сум, декартового добутку, операцій САА та натуралізації.

Доведення. Нехай, $P_1=(\Pi,\Delta_1,\delta_1)$, $P_2=(\Pi,\Delta_2,\delta_2)$ та $P_3=(\Pi,\Delta_3,\delta_3)$ - довільні процедури в просторі $\Pi=(T,S,S_0)$. Нехай для простоти $S_0=S$ (це не обмежує доведення). В прямих сумах стани процедур і P_1,P_2 та P_3 - будуть індексуватися індексами 1,2 та 3 відповідно.

Для перших трьох операцій твердження теореми випливає безпосередньо з визначень відповідних процедур.

Нехай $h=P_2^*\circ P_3^*$. Покажемо, як обчислити значення функції $h(s)$ за допомогою наступної квазі-прямої суми процедур $P_4=P_2^*\hat{\oplus}P_3^*=(\bar{\Pi}_4,\bar{\Delta}_2\cup\bar{\Delta}_3\cup\{\varphi_{23}\},\delta_4)$ в обчислювальному просторі $\bar{\Pi}_4=(T,S\oplus S,(S)_2)$. Спочатку P_4 буде працювати як перша компонента прямої суми $P_2\oplus P_3$ на початковому стані $(s)_2$ і зупиниться з певним результатом $s'\in(S)_2$. Потім операторна функція $\delta_4(s')=\varphi_{23}$ передасть керування другій компоненті $P_2\oplus P_3$, яка проведе обчислення на початковому стані $\varphi_{23}(s')$ і зупиниться з результатом $s''\in(F_3)_3$. До процедури P_4 залишилося додати не індексовані стани з S, інтерфейсні функції $\varphi_{02}:S\to(S)_2,\varphi_{30}:(S)_3\to S$, якими буде відповідно розпочинатися і закінчуватися повне обчислення значення $h(s)$ в процедурі P_5, де $P_5=(\Pi_5,\bar{\Delta}_1\cup\bar{\Delta}_2\cup\{\varphi_{02},\varphi_{23}\varphi_{30}\},\delta_5)$, $\Pi_5=(T,S\cup S\oplus S,S)$, $\delta_5(s)=\varphi_{02},s\in S,\delta_5(s')=\varphi_{23},\delta_5(s'')=\varphi_{30},\bar{\delta}_2(s')=\varnothing,\bar{\delta}_3(s'')=\varnothing$. На решті станів $\delta_5(s)=\delta_4(s)$. За побудовою, $P_5^*=P_2^*\circ P_3^*$.

Нехай $h=\{p(P_1^*)\to P_2^*\mid P_3^*\}$. Розглянемо обчислювальний простір $\Pi'=(T,S\cup S^3,S)$ і покажемо, що функція h є обчислювальною в обчислювальному просторі Π'. Розглянемо можливу схему функціонування відповідної процедури $P_4=(\Pi',\Delta_4,\delta_4)$. Спочатку вхідний стан s процедура за допомогою елементарної операції $\psi_1:S\to S^3$ переводить у робочий стан (s,s,s)

. Далі вона буде працювати як декартовий добуток $P_1 \times P_2 \times P_3$. Коли зупиниться процедура P_1, то в залежності від значення предикату p на її результаті, процедура P_4 у разі необхідності продовжить обчислення як P_2 або P_3 до їхньої зупинки. Елементарна операція $\psi_2 : S^3 \to S$ поверне актуальну компоненту (другу чи третю), яка і буде збігатися зі значенням $h(s)$. Формально це виглядає наступним чином. $\Delta_4 = \Delta_1 \times \Delta_2 \times \Delta_3 \cup \{\psi_1, \psi_2\}$, де $\psi_2(s_1, s_2, s_3) = s_2$, якщо $ps_1 = 1$ і $\psi_2(s_1, s_2, s_3) = s_3$, якщо $ps_1 = 0$. $F_4 = p^+(F_1) \times F_2 \times S_3 \cup p^-(F_1) \times S \times F_3$. Формальне визначення оперативної функції δ_4 винесено до Вправи 9. За побудовою, $P_4^* = h$.

Візьмемо операцію ітерації і покладемо $h = \{p(P_1^*) \to P_2^*\}$.Візьмемо обчислювальний простір $\Pi' = (T, S \cup S_1 \times S_2, S)$ і опишемо схему роботи процедури $P_4 = (\Pi', \Delta_4, \delta_4, p^+(F_2))$ для обчислення значенням $h(s)$. Спочатку вхідний стан s процедура за допомогою елементарної операції $\psi_1 : S \to \bar{S}_1 \times \bar{S}_2$ переводить у робочий стан $(\varphi_1 s, \varphi_2 s)$. Далі процедура P_4 по черзі працює з першою компонентою як процедура $\bar{P}_1 \times I$ до її зупинки, а потім – так само з другою компонентою як $I \times \bar{P}_2$, при умові, що значення предикату p на результаті роботи P_1 буде 0. При поверненні до роботи процедури $\bar{P}_1 \times I$ початковим станом буде стан $(\bar{q}, \bar{q})$, де $\bar{q}$ - заключний стан, на якому перед цим зупинилася процедура $\bar{P}_2$. Формальне визначення оперативної функції δ_4 винесено до Вправи 12. За побудовою, $P_4^* = h$.

Нехай $h = (P_1^*)_N$. Див. Вправу 14.)◄

Контрольні запитання та вправи

1. Що таке гомоморфізм процедур ?
2. Що таке ізморфні процедури ?
3. Що таке фактор –процедура?
4. Що таке вільна процедура ?
5. Дати визначення системи алгоритмічних алгебр.

6. Дати визначення алгебри Чорча та класів функцій: ПРФ, РФ, ЧРФ та класів примітивно рекурсивних, рекурсивних та частково-рекурсивних множин.
7. Довести, що клас унарних ЧРФ замкнений відносно операцій розширеної САА.
8. У чому полягає різниця між прямою і квазі-прямою сумами процедур?
9. Дати визначення прямої суми та прямого добутку процедур.
10. Задати оперативну функцію для декартового добутку процедур.
11. Задати оперативну функцію для розгалуження процедур з доведення теореми про замкненість.
12. Задати оперативну функцію для ітерації процедури з доведення теореми про замкненість.
13. Довести замкненість класу процедурно обчислювальних функцій відносно ітерації, використавши операцію квазі-прямої суми.
14. Довести, що клас функцій, обчислюваних процедурами, замкнений відносно операції натуралізації.

Розділ 4. Традиційні моделі алгоритмів в процедурній платформі

Як випливає з леми 2.1, всі традиційні алгоритмічні системи та числення мають свої процедурні аналоги. З'ясуємо їхнє місце в процедурній платформі.

Зазначимо, що в класичній теорії алгоритмів розрізнюють власне алгоритмічні системи та числення. Поняття А-процедури дозволяє їх концептуально об'єднати.

Щоб визначити вид А-процедури для конкретного класу алгоритмів або числення необхідно: 1)задати обчислювальний простір, 2) задати сукупність елементарних операцій на станах, 3)визначити вхідні стани, а за необхідності і вихідні та функцію-проекцію і головне – 4) визначити вид розв'язності оперативної таблиці.

Якщо звернутися до існуючих моделей алгоритмів та числень, то можна помітити таку закономірність: 1) вони, як процедури є в основному автоматними (за винятком більшості числень), тобто переходи в обчисленнях залежать тільки від останнього стану обчислення і не залежать від попередніх станів чи номеру кроку обчислення і 2) для цих переходів визначальними є стани не в усьому їхньому об'ємі, а тільки вмісти певних обмежених (активних за [10, 11]) їхніх структурних фрагментів. Збіг вмістів цих обмежених фрагментів лежить в основі ядерної еквівалентності і скінченності синтаксичних оперативних таблиць.

Розглянемо спочатку не представницькі класи формальних алгоритмів та числень[8].

1.Канонічне алгебраїчне числення регулярної алгебри мов [12,13]. Нехай X – довільний скінченний алфавіт, $B(X)$ булеан всіх підмножин слів в цьому алфавіті. Регулярною алгеброю над алфавітом X називається алгебра $A = (B(X); \cup, \circ, ^*)$ з операціями теоретико-множинним об'єднанням, конкатенацією та ітерації відповідно. Конкатенація двох мов визначається як мова $L_1 \circ L_2 = \{uv : u \in L_1, v \in L_2\}$, а ітерація мови L^* - як нескінченне об'єднання мов $\bigcup_{n=0}^{\infty} L^n$, таке що $L^0 = \{\varepsilon\}, L^n = L^{n-1} \circ L$. Елементи підалгебри алгебри A з породжувальними елементами: $\varnothing$- порожньою мовою і

[8] Представницький клас формальних алгоритмів обчислює всі ЧРФ.

одноелементними мовами $\{x\}, x \in X$, називаються *регулярними мовами* в алфавіті X. За побудовою, теоремами канонічного алгебричного числення регулярної алгебри мов є регулярні мови в алфавіті X і тільки вони. Це числення є алгоритмічним, тому що всі операції регулярної алгебри мов є конструктивними.

2.Скінченні X-автомати (СА) [12,13]. Скінченні X-автомати призначені для розпізнавання регулярних мов в скінченному алфавіті X. Автомат має вхідну стрічку, в яку записується слово для аналізу. В процесі обчислення він не змінює це слово, а тільки «робить висновок» після зупинки обчислення чи належить воно даній регулярній мові чи ні. Стани процедури СА мають структуру пар $(q,u|v)$, де q належить певній скінченній сукупності Q *внутрішніх* станів автомата, u,v – це слова у вхідному алфавіті X, а символ $|\notin X$ є роздільником. Запис $u|v$, де $u,v \in X^*$ означає, що вхідне словом, яке аналізується збігається з uv, вже оброблено його префікс u і доступним наступним символом є перший символ за роздільником, тобто перший символ слова v. Перебуваючи в стані $(q_t,u_t|av_t)$ з доступним символом a, обчислення згідно функції переходів автомата вигляду $\delta: Q\times(X\cup\varepsilon)\to 2^Q$ переходить у новий стан, в якому $u_{t+1}=u_t, v_{t+1}=v_t$. Як бачимо, вхідне слово в процесі обчислення дійсно не змінюється, може змінитися тільки доступний символ – ним стає наступний справа, тобто перший символ слова v_{t+1}, якщо це слово не порожнє. Можливий також ε–перехід, при якому змінюється тільки внутрішній стан автомата, а доступний символ не змінюється. Обчислення закінчується, коли на вході вичерпуються доступні символи. При цьому, якщо воно, розпочавшись з вхідної конфігурації $(q_0,|v)$, закінчується в одному із заключних станів вигляду $(q,v|)$, $q\in F$, то говорять, що слово v *допускається* автоматом. Може допускатися і префікс вхідного слова, якщо обчислення на ньому переходить в заключний внутрішній стан. Тобто в скінченних X–автоматах мають місце гіперобчислення. Сукупності всіх слів, які допускаються автоматом A називається *мовою, що допускається* цим автоматом і позначається $L(A)$.

Ядерна еквівалентність оперативної функції процедури СА дуже проста й чисто синтаксична: довільні стани $(q,u|av)$ та $(q',u'|a'v')$ є еквівалентними, якщо $q=q'$ і $a=a'$. Їй відповідає скінченна оперативна

таблиця, в першому стовпчику якої розташовані пари $(q,a)\in Q\times(X\cup\varepsilon)$ з області визначення функції переходів автомата δ, а в другому відповідно її значення. Таку таблицю простіше задати іншою скінченною таблицею - синтаксичною, рядки якої відповідають внутрішнім станам, стовпчики символам алфавіту $X\cup\varepsilon$, а на перетині рядків і стовпчиків – розміщено значення функції переходів. Отже, процедура СА є конструктивною та автоматною.

Залишилося визначити проективну функцію $\mu: X^*\to\{0,1\}$, яку обчислює А-процедура. Вона відповідає функції-проекції $pr(q_0,|v)=v$, $pr(q,v|)=1$ для $q\in F$ і $pr(q,v|)=0$ у супротивному. Функція μ збігається з характеристичною функцією мови, яку допускає автомат A. Скінченні автомати допускають клас регулярних мов в алфавіті X.

3.Скінченні детерміновані $X-Y-$автомати [8,13]. Нехай X,Y- довільні вхідний та вихідний алфавіти. Скінченні $X-Y-$автомати призначені для автоматних перетворень регулярних мов вигляду $f: X^*\to Y^*$. Автомат A має дві стрічки – вхідну і вихідну і в процесі обчислення посимвольно перетворює вхідне слово у вихідне. Стани процедури $X-Y-$ автомата мають структуру трійок $(q,u|v,w)$, де q належить певній скінченній сукупності Q *внутрішніх* станів, u,v – це слова у вхідному алфавіті X, а символ $|\notin X$ є роздільником, $w\in Y^*$ - слово у вихідному алфавіті Y. Запис $u|v$, де $u,v\in X^*$ означає, що словом, яке перетворюється є uv, процедура вже прочитала його префікс u і перетворив у слово wі доступним наступним символом вхідного слова є перший символ за роздільником, тобто перший символ підслова v. Перебуваючи в стані $(q_t,u_t|av_t,w)$ і читаючи символ a, процедура згідно його функції переходів переходить у новий стан $(q_{t+1},u_{t+1}a|v_{t+1},wy)$, в якому $u_{t+1}=u_t, v_{t+1}=v_t$.

Як бачимо, вхідне слово процедура в процесі його роботи не змінюється, може змінюватися тільки доступний символ – ним стає наступний за попереднім доступним справа, разом з тим до вихідного слова дописано справа слово $y\in Y^*$. Процедура закінчує свою роботу, коли на вході вичерпуються доступні символи. Якщо процедура, розпочавши свою роботу з вхідної конфігурації $(q_0,|v,\varepsilon)$, закінчує її в одному із заключних станів вигляду $(q,v|,w)$, $q\in F$, то говорять, що слово v перетворюється процедурою

у слово w. На відміну від скінченних $X-$автоматів, обчислення в $X-Y-$ автоматах є детермінованим і не містить ε-переходів. Процедура завжди зупиняється при переході в заключний стан.

Формально функція переходів $X-Y-$автомата має вигляд $\delta: Q\times X \to Q\times Y^*$.

$X-Y-$автомат A називається автоматом Мілі, якщо його функція переходів має вигляд $\delta: Q\times X \to Q\times Y$. Автомат Мілі називається автоматом Мура, якщо для будь-яких $q, p_1, p_2 \in Q, x_1, x_2 \in X, y_1, y_2 \in Y$ з того, що $\delta(q, x_1) = (p_1, y_1)$ і $\delta(q, x_2) = (p_2, y_2)$ випливає $y_1 = y_2$. В автоматі Мура вихідний сигнал y не залежить від вхідного сигналу x на вході. Тому в цих автоматах функцію переходів розділяють на дві: $\delta: Q\times X \to Q$ і $\mu: Q \to Y$ і автомат Мура – це семірка $A = (Q, X, Y, \delta, \mu, q_0, F)$.

Ядерна еквівалентність оперативної функції процедури $X-Y-$ автомата та ж, що і для СА – чисто синтаксична: довільні стани $(q, u \mid av, w)$ та $(q', u' \mid a'v', w')$ є еквівалентними, якщо $q = q'$ і $a = a'$. Їй відповідає оперативна таблиця, в першому стовпчику якою розташовані пари $(q, a) \in Q\times X$ з області визначення функцій переходів, а в другому відповідно її значення. Таку таблицю зручніше задати, як і у випадку $X-$автоматів, іншою таблицею - синтаксичною, рядки якої відповідають внутрішнім станам, стовпчики символам алфавіту X, а на перетині рядків і стовпчиків – розміщено значення функції переходів. Таблиця скінченна, тому процедура $X-Y-$ автомата є конструктивною.

Розглянемо приклад $X-Y-$автомата. Нехай символ c - роздільник довільного слова $u \in \{a, b, c\}^*$ на лексеми, до яких відносяться усі підслова в алфавіті $\{a, b\}$ між двома роздільниками та підслова до першого і після останнього входження роздільника. Наприклад, у слові $u = abccaca$ лексемами є підслова ab, a, a .

Наступний $X-Y-$автомат $A_2 = (Q, X, Y, \delta, q_0, F),$ вилучає з вхідного слова усі «зайві» роздільники. $Q = \{q_0, q_1, q_2\}$ $X = \{a, b, c\}$ $Y = \{a, b, c\}$, , $\delta = \{q_0 c \to q_0 / \varepsilon, q_0 a \to q_1 / a, q_0 b \to q_1 / b, q_1 c \to q_2 / \varepsilon, q_2 c \to q_2 / \varepsilon, q_1 a \to q_1 / a, q_1 b \to q_1 / b,$, $q_2 a \to q_1 / ca, q_2 b \to q_1 / cb\}$ $F = Q$. У стані q_0 автомат вилучає всі роздільники на початку слова, в станах q_1 та q_2 - в середині слова та його кінці, а в стані q_2 відновлюється один роздільник. Його СОТ має вигляд:

	a	b	c
q_0	q_1 / a	q_1 / b	q_0 / ε
q_1	q_1 / a	q_1 / b	q_2 / ε
q_2	q_1 / ca	q_1 / cb	q_2 / ε

4.Автомати з магазинною[9] пам'яттю (МП-автомати) [15]. Від скінченного X – автомата МП-автомат відрізняється наявністю в структурі його станів третього компонента – стеку символів, який служить для зберігання додаткової внутрішньої інформації. Перші два компоненти функція переходу МП-автомата змінює так, як і у випадку скінченних автоматів, а третю – за допомогою стекових операцій *pop* (зняти) і *push* (додати елемент в стек).

МП-автомат –це сімка $A=(Q, X, Y, \delta, q_0, Z_0, F),$, де:

1) Q – скінченна множина внутрішніх станів;
2) X – вхідний алфавіт;
3) Y – алфавіт магазинних символів;
4) $\delta : Q \times (X \cup \varepsilon) \times Y \to Q \times Y^*$ – функція переходів;
5) q_0 – початковий внутрішній стан;
6) Z_0 – початковий вміст стеку;
7) $F \subseteq Q$ – множина внутрішніх заключних станів.

Функція переходів МП може бути багатозначною, а це значить, що відповідний процедурний алгоритм може бути недетермінований. Стани процедури МП мають структуру трійок $(q, u \mid v, wZ)$, де q належить певній скінченній сукупності Q , так званих, *внутрішніх* станів, u, v – це слова у вхідному алфавіті X , а символ $\mid \notin X$ є, як і раніше, роздільником, $wZ \in Y^*$ - слово в магазині, останнім символом якого є Z. Запис $u \mid v$, де $u, v \in X^*$ означає, що словом, яке обробляється є uv, процедура вже прочитала його префікс u і наступним доступним символом вхідного слова є перший

[9] Магазин є синонімом терміну стек - структури даних, яка зберігає послідовності однотипних елементів з єдиним доступним елементом - останнім.

символ за роздільником. Перебуваючи в стані $(q_t, u_t \mid av_t, wZ)$ і читаючи символ a на вході і символ Z в магазині, процедура згідно функції переходів автомату переходить у новий стан $(q_{t+1}, u_{t+1}a \mid v_{t+1}, wy)$, в якому $u_{t+1} = u_t, v_{t+1} = v_t$. При цьому $y = \varepsilon$ відповідає операції *pop* в магазині, $y \neq \varepsilon$ - операції *push*.

Як бачимо, вхідне слово в процесі обчислення не змінюється, може змінюватися тільки його доступний символ – ним може стати наступний за попереднім доступним справа, разом з тим в магазині доступний символ замінюється на слово y, останній символ якого у випадку операції *push* стає наступним доступним у магазині.

Як і у випадку СА можливі також ε – переходи, при яких доступний символ на вході не змінюється**.** Процедура закінчує обчислення, коли на вході вичерпуються доступні символи. Якщо вона, розпочавши обчислення з вхідного стану $(q_0, \mid v, Z_0)$, закінчує його в одному із заключних станів вигляду $(q, v \mid, w)$**,** $q \in F$**,** то говорять, що слово v допускається автоматом. В деяких випадках заключні стани вимагають спустошення магазину. За означенням, з порожнім стеком процедура продовжувати обчислення не може.

Ядерна еквівалентність оперативної функції процедури МП-автомата аналогічна тій, що і для СА – вона чисто синтаксична: довільні стани $(q, u \mid av, wZ)$ та $(q', u' \mid a'v', w'Z')$ є еквівалентними, якщо $q = q'$, $a = a'$ та $Z = Z'$. В СОТ рядки відповідають парам $(q, a) \in Q \times (X \cup \varepsilon)$, стовпчики - символам магазинного алфавіту $Z \in Y$, а на перетині рядків і стовпчиків – розміщено значення функцій переходів $\delta(q, a, Z)$. Таблиця скінченна, тому процедура МП-автомата є конструктивною і автоматною.

Проективна функцію $\mu : X^* \to \{0,1\}$, яку обчислює А-процедура, відповідає функції-проекції $pr(q_0, \mid v, Z_0) = v$ і $pr(q, v \mid, w) = 1$ для $q \in F$ та $pr(q, v \mid, w) = 0$ - у супротивному. Для обчислень зі спустошенням магазину необхідно покласти $w = \varepsilon$. Функція автомата μ є характеристичною функцією мови $L(A)$, яку допускає МП-автомат A.

5.Контекстно-вільні граматики [15,16]. Контекстно-вільні граматики (КВ-граматики) – це клас *A*-числень, призначених для математичного опису формальних мов, які допускаються МП-автоматами та їхнього синтаксичного аналізу. *КВ-граматикою* називається четвірка $G = (N, T, P, S)$,

де N та T – алфавіти нетермінальних та термінальних символів, P – множина продукцій вигляду $X \to w$, де $X \in N$, $w \in (N \cup T)^*$, S – аксіома з множини N. Слово uwv безпосередньо виводиться зі слова uXv в граматиці G (позначається $uXv \to uwv$), якщо $X \to w \in P$.

Позначимо $\Rightarrow$ рефлексивно-транзитивне замкнення відношення безпосереднього виведення граматики G. *Контекстно-вільною мовою* (*КВ-мовою*), породженою граматикою G, називається множина слів $L(G) = \{u \in T^* : S \Rightarrow u\}$.

КВ-граматиці відповідає A-числення зі станами – словами в об'єднаному алфавіті $(T \cup N)$, аксіомою S, правилами виведення – підстановками замість довільного входження нетерміналу в слово-стан правої частини відповідної продукції, заключними станами є слова в термінальному алфавіті T. Числення є недетермінованим і автоматним. Для підстановки може бути вибране будь-яке входження будь-якого нетерміналу.

Ядерна еквівалентність оперативної функції КВ-граматики очевидна: два стани u та v еквівалентні, якщо в них одна і та ж сукупність нетерміналів (їхніх входжень може бути різна кількість). СОТ процедури є скінченною: у першому стовпчику розташовані усі підмножини нетермінальних символів, у другому сукупність правих частин продукцій для нетерміналів підмножини.

Покладемо u^R - дзеркальне відображення слова u. Наступна граматика $G_1 = (\{S\}, (\{a,b\}, \{S \to aSa \mid bSb \mid \varepsilon \mid a \mid b\}, S)$, де метасимвол $\mid$ розділяє альтернативні праві частини продукцій з нетерміналом S зліва, породжує мову усіх дзеркальних слів в алфавіті $\{a,b\}$. $L(G_1) = \{uu^R : u \in \{a,b\}^*\} \cup \{uau^R : u \in \{a,b\}^*\} \cup \{ubu^R : u \in \{a,b\}^*\}$. СОТ А-числення граматики G_1 має вигляд:

Нетермінал	Праві частини правил
S	$\{aSa, bSb, \varepsilon, a, b\}$

КВ-граматики є частковим випадком формальних граматик [16], в яких правила виведення мають вигляд $u \to w \in P$, $u \in (T \cup N)^*$. Формальні

граматики, на відміну від КВ –граматик, являють собою вже представницький клас числень

6.Пропозиційні числення (ПЧ). [1] Це повні числення класичної пропозиційної логіки, тобто такі, що їхня теорія збігаються з множиною всіх формул-тавтологій. Повних ПЧ відомо багато. Наступне є одним з них. Пропозиційними формулами є пропозиційні змінні та формули, побудовані з них за допомогою зв'язок заперечення $\neg$, диз'юнкції $\vee$ та круглих дужок (,). Аксіомами є формули $\neg A \vee A$, правилами виведення:

П1 $A \vdash B \vee A$ (правило розширення),

П2 $A \vee A \vdash A$ (правило поглинання),

П3 $A \vee (B \vee C) \vdash (A \vee B) \vee C$ (права асоціативність),

П4 $A \vee B, \neg A \vee C \vdash B \vee C$ (правило перетину).

Станами A-числення ПЧ є формули, аксіоми і правила виведення – наведені вище. Це не автоматне числення. Його теорія збігається з множиною всіх тавтологій. СОТ A-числення можна задати у вигляді:

Вигляд префіксу	Результат правила виведення
Містить стан A	$B \vee A$
Містить стан $A \vee A$	A
Містить стан $A \vee (B \vee C)$	$(A \vee B) \vee C$
Містить стани $A \vee B, \neg A \vee C$	$B \vee C$

Насправді, якщо бути точним, то два префікси є ядерно еквівалентними, якщо їм відповідає одна й та сукупність застосовних до них правил П1-П4. Тоді в першому стовпчику потрібно перерахувати всі комбінації лівих частин правил, а в другому – навести всі результати їхніх застосувань. Така таблиця теж буде скінченною.

Тепер розглянемо основні представницькі класи алгоритмів та числень.

6. Машини Коломогорова-Успенського (МКУ). [9,10] Метою авторів було сформулювати якомога більш загальний представницький клас

алгоритмів. Ідея базувалась на спробі уточнення поняття констуктивного об'єкту як спеціального зв'язного орінтованого графу - (Б,*к*)-комплекса, ансамблів таких об'єктів та локальних операцій над ними. Надалі, (Б,*к*)-комплекси будемо називати станами. В локальних операціях над довільним станом «активна» метрично обмежена його частина замінюється іншою. Кожна така операція має вигляд $\alpha \to \beta$, де ліва частина операції задає активну частину довільного стану, а права – ту, яка її заміщує в процесі операції. МКУ Γ задається як скінченний набір локальних операцій $\{\alpha_1 \to \beta_1, \alpha_2 \to \beta_2, ..., \alpha_k \to \beta_k\}$, з попарно різними лівими частинами, з множинами вхідних X та вихідних Y станів.

Обчислення за машиною Γ визначається як послідовність станів $s_0 ... s_n, ...$, в якому $s_0 \in X$ і кожний наступний стан s_n є результатом застосування до стану s_{n-1} тієї єдиної потенційно застосовної до нього локальної операції машини.

В машинах Колмогорова механізм завершення обчислення базується на невизначеності функції переходів. Машина обчислює певну функцію $\Gamma^*: X \to Y$ або певний проективний її варіант. *A*-процедура МКУ очевидна як і ядерна еквівалентність її оперативної функції: два стани машини є еквівалентними, якщо їхні «активні» частини збігаються. Структура цих частин і є синтаксичною ознакою відповідного класу еквівалентності. Нехай $S_1, ..., S_к$ - всі ознаки ядерної еквівалентних станів. СОТ МКУ має вигляд:

Клас	Локальна операція
S_1	$\alpha_1 \to \beta_1,$
S_2	$\alpha_2 \to \beta_2,$
…	…
$S_к$	$\alpha_к \to \beta_к,$

Розглянуті вище класи автоматів і наступний є частковими випадками машин Колмогорова-Успенського

7. Машини Тюрінга (МТ) [17,16, 1]**.** Це один з перших представницьких класів формальних алгоритмів. МТ призначені для

перетворень слів в алфавіті X. Їх можна розглядати як скінченний X-автомат з додатковими можливостями. Він теж має вхідну стрічку, в якій розміщується вхідне слово, але на відміну від скінченного X-автомата, в процесі обчислення може замінювати доступний символ на інший з вхідного алфавіту і робити наступним доступним не тільки символ справа, а й зліва. МТ завжди зупиняється при переході в заключний стан (відсутні гіперобчислення).

Стани МТ мають структуру пар $(q, uc|av)$, де q є внутрішній стан зі скінченної сукупності Q таких станів, u,v –слова у вхідному алфавіті X, $c,a \in X$, а символ $| \notin X$, як і раніше, є роздільником. Запис $uc|av$, означає, що вхідне слово перетворено в слово $ucav$ і доступним символом є a. Перебуваючи в стані $(q_t, u_t c|av_t)$ і читаючи символ a, автомат згідно його функції переходів переходить у новий внутрішній стан q_{t+1}, доступний символ a замінює на b, який може залишити доступним і далі або зробити доступним символ зліва від нього c чи наступний за ним справа – перший у слові v_t.

У першому, другому та третьому випадках новими станами МТ будуть відповідно $(q_{t+1}, u_t c|bv_t)$ $(q_{t+1}, u_t|cbv_t)$ та $(q_t, u_t cb|v_t)$. Якщо МТ розпочинає свою роботу з вхідної конфігурації $(q_0, |v)$ і закінчує її в заключному стані вигляду $(q^*, w|u)$, де q^* - виділений заключний внутрішній стан, то говорять, що слово v перетворюється МТ у слово wu.

Формально МТ - це п'ятірка $A = (Q, X, \delta, q_0, q^*)$, де:

Q – скінченна множина внутрішніх станів;

X – скінченна множина вхідних символів (вхідний алфавіт);

$\delta : Q \times X \to Q \times X \times \{L, R, \varepsilon\}$ – скінченна багатозначна *функція переходів*, яка керує роботою МТ;

$q_0 \in Q$ – початковий внутрішній стан;

$q^* \in Q$ – заключний стан.

L, R та ε означають, що в процесі обчислення новим доступним символом на вході є відповідно перший зліва, справа та доступний символ не змінюється. Якщо функція переходів МТ однозначна, то машина називається *детермінованою.*

Ядерна еквівалентність оперативної функції процедури МТ та ж, що і у випадку СА: довільні стани $(q,u\,|\,av)$ та $(q',u'\,|\,a'v')$ є еквівалентними, якщо $q=q'$ і $a=a'$. Їй відповідає скінченна СОТ, в першому стовпчику якою розташовані пари $(q,a)\in Q\times X$ з області визначення функції переходів МТ, а в другому відповідно її значення. Таку таблицю, як і для СА, простіше задати іншою таблицею, рядки якої відповідають внутрішнім станам, стовпчики символам алфавіту X, а на перетині рядків і стовпчиків – розміщено значення функції переходів.

Проективну функцію $A^*: X^* \to X^*$, яку обчислює МТ, задає функція-проекція $pr(q_0,|\,v)=v$, $pr(q^*,u\,|\,v)=uv$.

Аналогічно можна розглянути МТ з кількома стрічками і МТ з кількома

доступними комірками (з кількома головками) [6].

8. Нормальні алгоритми Маркова (НА). [18,1,4] Нормальним алгоритмом називається трійка $A=(N,X,P)$, де N,X - скінченні алфавіти відповідно допоміжних і термінальних символів, P - упорядкована сукупність правил вигляду $u_i \to v_i$ або $u_i \to v_i\bullet$, де $u_i, v_i \in (N\cup T)^*, i=\overline{1,n}$, символ $\bullet \notin N\cup X$ і є службовим. Правила з крапкою називаються заключними, права його частина не містить допоміжних символів. Правило $u_i \to v_i$ $(u_i \mapsto v_i)$ задає елементарну операцію на словах в об'єднаному алфавіті $N\cup X$, яка полягає в заміні першого входження в слові лівої частини правила на його праву частину. Якщо такого входження немає, то правило не застосовне до даного слова.

Станами A-процедури НА є слова в об'єднаному алфавіті $N\cup X$, вхідними й заключними станами – всі слова в термінальному алфавіті. A-процедура є автоматною, її оперативна функція δ на стані повертає найменший номер застосовного до нього правила з P. Обчислення зупиняється тільки після появі у стані символа $\bullet$ або коли жодне з правил сукупності P не застосовне до поточного слова. В обох випадках результатом обчислення є останній його стан при умові, що він не містить допоміжних символів і з якого вилучена крапка. Таким чином, НА A обчислює певну словарну функцію $A^*: X^* \to X^*$.

Класи ядерної еквівалентності оперативної функції *A*-процедури НА $S_1,...,S_n$ є розв'язними, відповідний алгоритм полягає в з'ясуванні найменшого номера правила, яке застосовне до даного слова. Оперативна таблиця має вигляд:

Клас	Операція
S_1	$u_1 \to v_1$,
S_2	$u_2 \to v_2$,
…	…
S_n	$u_n \to v_n$,

9. **Регістрові машини (РМ).** [19,20,1,4] РМ – абстрактна модель комп'ютера. В моделі відсутня оперативна пам'ять, але є потенційно не обмежена кількість числових регістрів $R_0, R_1,..., R_m,...$ Нехай $I_1 I_2 ... I_n$ - послідовність машинних команд, яка складає програму РМ. Станами *A*-алгоритму регістрової машини виступає сукупність всіх її регістрів разом з їхніми вмістами, серед яких є лічильник команд *PC*. Останній містить адресу команди, яка буде виконуватись на наступному кроці обчислення РМ. В початковому стані *PC*=1, в заключних станах - *PC*>*n*, де *n* – адреса останньої машинної команди в програмі. *A*-процедура – автоматна. Після виконання програми функція-проекція *pr* повертає вміст нульового регістру заключного стану. Два стани *A*-алгоритму - ядерно еквівалентні, якщо в них значення лічильника команд *PC* збігаються. Тоді СОТ *A*-алгоритму має вигляд:

Вміст *PC*	Операція
1	I_1
2	I_2
…	…
n	I_n

]

Варіантом РМ є операторні алгоритми [20**].**

10. Дискретні перетворювачі Глушкова (ДП) [14,8]. ДП є комбінацією двох автоматів – керуючого $X-Y$ –автомата Мілі $A=(Q,X,Y,\delta_A,q_0,q^*)$ і оперативного $Y-X$ –автомата Мура $B=(P,Y,X,\delta_B,\mu,P_0,F)$ з виділеною підмножиною вхідних станів $P_0\subseteq P$, спри цьому множина станів P не обов'язково скінченна.

В ДП вихідні сигнали керуючого автомата посилаються на вхід оперативного, а вихідні оперативного – на вхід керуючого. ДП обчислює відображення $f:P_0\to P$ наступним чином. Керуючий автомат на нульовому кроці обчислення перебуває в стані q_0 , а оперативний - у певному стані $p_0\in P_0$. В процесі обчислення значення $f(p_0)$ породжуються послідовності: $x_1,x_2,...;\ y_1,y_2,...;\ q_0,q_1,...;\ p_0,p_1,...,$ де $x_i=\mu(p_{i-1}),\ \delta_A(q_{i-1},x_i)=(q_i,y_i)),$ $p_i=\delta_B(p_{i-1},y_i),\ i=1,2,..$ (*) . Процес припиняється на $n-$му кроці, якщо керуючий автомат переходить в заключний стан q^* або одна з функцій автоматів на цьому кроці не визначена і $q_n\neq q^*$. У першому випадку $f(p_0)=p_n$, а в другому і коли процес обчислення нескінченний значення $f(p_0)$ не визначено.

Станами процедури ДП виступають четвірки $X\times Q\times Y\times P$ (вхідний сигнал, внутрішній стан керуючого автомата, вихідний сигнал, внутрішній стан оперативного автомата). В початковому стані процедури внутрішній стан керуючого автомата – початковий q_0, стан оперативного автомату – довільний $p_0\in P_0$, початкові вхідний і вихідний сигнали функціонально залежать від початкового стану оперативного автомата. В заключних станах – стан керуючого автомата – заключний q^*. Функція-проекція ДП *pr* повертає стан оперативного автомата. Позначимо Ω елементарну операцію, яка поточний стан перетворює в наступний згідно співвідношенням (*). Це буде єдина елементарна операція в процедурі ДП. Враховуючи, що СА Мілі дискретного перетворювача задається скінченною таблицею, то алгоритмічність процедури ДП залежить виключно від наявності скінченного опису його автомата Мура.

Як приклад, розглянемо як подати МТ $A=(Q,X,\delta,q_0,q^*)$ у вигляді ДП. Керуючий автомат Мілі буде приймати від оперативного автомата

інформацію про доступний символ на стрічці і повертати йому інформацію про те, як змінити стан стрічки, а оперативний автомат Мура реалізую зміну стану стрічки МТ. Покладемо вихідний алфавіт автомата Мілі $Y = X \times \{L, R, \varepsilon\}$ і сукупність внутрішніх станів $P_1 = \{u \mid v : u, v \in X^*, \mid \notin X\}$ автомата Мура. Автоматом Мілі буде $X - Y$ – автомат $A_1 = (Q, X, Y, \delta_1, q_0, q^*)$, де $\delta_1(q,a) = (p,(b,t))$, якщо $\delta(q,a) = (p,b,t)$, а $Y - X -$ автоматом Мура автомат $B_1 = (P_1, Y, X, \delta_{B_1}, \mu_1, \{\mid v : v \in X^*\}, \varnothing)$, де $\delta_{B_1}(u \mid av, (b,t)) = u' \mid a'v'$ - стан стрічки МТ після виконання в ній МТ переходу $\delta(q,a) = (p,b,t)$, $\mu_1(u \mid av) = a$. Функції δ_{B_1} та μ_1 описуються скінченними таблицями, тому відповідна процедура ДП буде конструктивною.

11. Канонічне алгебраїчне числення систем алгоритмічних алгебр Глушкова (САА) [14,8]. Використовуються для аналізу та синтезу дискретних перетворювачів. Нагадаємо що таке САА. Нехай $\sigma = \{f_1, ..., f_n, ...; p_1, ..., p_m, ...\}$ - довільна унарна сигнатура функціональних та предикатних символів $A = (A, \sigma_A)$ [10] - довільна алгебраїчна система сигнатури σ. Нехай та $F^1(A)$ та $P^1(A)$ множини всіх відображень типів $A \to A$ та $A \to \{0,1\}$ відповідно. Розглянемо двоосновну АС $A = (F^1(A), P^1(A); \circ, (\to\mid), \{\to\})$ з операціями множення, розгалуження та ітерації, які вже були визначені в Розд. 3. Підсистема системи A, породжена функціями і предикатами АС $A = (A, \sigma_A)$ називається системою алгоритмічних алгебр Глушкова. Елементи САА (тобто її функції і предикати) називаються регулярними в сигнатурі σ .

Регулярні функції та предикати в сигнатурі σ є теоремами канонічного алгебраїчного числення САА. Як і у випадку дискретних перетворювачів, це числення буде алгоритмічним при умові конструктивності АС $A = (A, \sigma_A)$.

Щоб уникнути введення в сигнатурі σ предикатних символів, їхню роль іноді покладають на звичайні алгебраїчні операції $p : S \to S$. При цьому роль істинністного значення 0 відводиться певній виділеній сигнатурній константі f^0, а істинністне значення 1 представляють всі інші елементи носія. Такий підхід має місце у мовах програмування та в програмних алгебрах.

[10] A позначає як саму АС, так і її носій, але з контексту завжди буде зрозуміло, що мається на увазі.

12. Канонічні алгебричні числення примітивних програмних алгебр (ППА)[1]. ППА - це узагальнення САА на випадок функцій з простими структурами даних на натуральних числах. Сюди ж відносяться й інші класи програмних алгебр - квазі-арних функцій, функцій над номінативними даними тощо [1].

13. Граф-схеми Калужніна (ГС) [21]**.** ГС є спеціальними зваженими бінарними графами, в яких виділено два вузли початковий і заключний. Вузли графа поділені на дві групи: вузли-розпізнавачі та вузли-перетворювачі. Із перших вузлів у графі виходить одна стрічка, із других – дві, відмічених символами «+» та «-» . Із заключного вузла стрілки не виходять і він може бути не зваженим. Решта вузлів зважені символами унарної сигнатури $\sigma = \{f_1,...,f_n; p_1,...,p_m\}$, при цьому вузлам-перетворювачам поставлені у відповідність функціональні символи сигнатури, вузлам–розпізнавачам – предикатні. Для інтерпретації ГС вибирається алебраїчна система $A = (A, \sigma_A)$. Обчислення $a_0, a_1,..., a_n,...$ за ГС розпочинається з початкової вершини і далі відбувається прохід по графу за стрілками. При проходженні вузла-перетворювача, зваженого символом f, в обчисленні з'являється новий стан $a_i = f_A(a_{i-1})$, де a_{i-1} - поточний на даний момент стан. При проходженні вершини-розпізнавача, зваженого символом p, вибирається стрілка з символом +, якщо $p_A(a_{i-1}) = 1$, де a_{i-1} - поточний на даний момент стан, і стрілка з символом -, у супротивному. Обчислення припиняється при потраплянні його в заключний вузол. Останній стан обчислення є його результатом.

Занумеруємо всі вузли ГС числами від 0 до *n,* де 0 – номер початкового вузла, *n* - заключного. Станами процедури інтерпретованої ГС виступають пари (номер вершини ГС, елемент предметної області *A*). У початковому стані перша компонента 0, у заключному – номер заключного вузла ГС. Процедура - автоматна. Проективна функція повертає другий компонент стану, тобто має тип $A \to A$. Ядерна еквівалентність оперативної функції така: два стани процедури ядерно еквівалентні, якщо їхні перші компоненти збігаються і а) ця компонента є номером вузла-перетворювача або б) ця компонента є вузлом з предикатним символом p і значення предикату p_A на других компонентах станів збігаються. Враховуючи, що синтаксично

ГС як скінченний граф є конструктивним об'єктом, то алгоритмічність процедури ГС залежить виключно від конструктивності алгебраїчної системи $A=(A,\sigma_A)$.

Прикладами ГС є численні класи схем програм та блок-схем алгоритмів такі, як схеми Янова, операторні схеми, структурні та стандартні схеми програм з пам'яттю та інші [22].

14. Розмічені транзитивні системи. [23,4] Якщо в автоматних унарних процедурах-численнях нівелювати вплив оперативної функції (тобто, коли ця функція на всіх станах повертає множину всіх елементарних операцій Δ), то отримаємо варіант розмічених транзиційних систем.

15. Арифметичне числення Пеано (АЧП). [17,1] Розглядається формальна арифметика з аксіоматикою Пеано. Відповідна їй процедура є *A*-численням, станами якого виступають арифметичні формули, правилами виведення – правила виведення АЧП, аксіоми *A*-численням - ті ж, що і в АЧП, а оперативна функція δ за префіксом обчислення повертає сукупність правил, застосовних до формул префікса і аксіом.

Два префікси ядерно еквівалентні, якщо до них застосовна ода і та ж сукупність правил виведення.

16. $\lambda-$**числення** [24,5]. Історично одне з перших представницьких формальних числень. Це числення рівностей $\lambda-$термів $t_1=t_2$, яке існує в декількох варіантах. Ми будемо розглядати алгоритмічний його варіант як числення просто $\lambda-$ термів. Дамо визначення $\lambda-$ терму. Нехай $V=\{v_0,v_1,...\}$ - певна сукупність змінних. У словах виду $\lambda x.t$ всі входження змінної x в t називаються зв'язаними. Входження змінної x в деяке слово u називаються вільним, якщо воно не є зв'язаними. Позначимо $FV(u)$ сукупність всіх змінних, які мають вільні входження в слово u**.** Базовими $\lambda-$термами є змінні сукупності V. Інші $\lambda-$терми будуються індуктивно за двома правилами: 1) Якщо t - $\lambda-$ терм і $x\in FV(t)$, то слово $\lambda x.t$ є $\lambda-$термом; 2) Якщо t_1 і t_2 - $\lambda-$ терми, то слово $(t_1\,t_2)$ є $\lambda-$термом. $\lambda-$терм вигляду $(\lambda x.t_1t_2)$ називається термом–застосуванням.

Станами відповідної *A*-процедури виступають $\lambda-$терми, вхідними станами - терми-застосування, заключними – терми, до яких до яких не застосовне правило редукції, яке є єдиною операцією А-процедури.

Позначимо $t_1[x:=t_2]$ результат підстановки терму t_2 в терм t_1 замість всіх вільних входжень змінної x. Правило редукції на $\lambda-$термі-застосуванні $(\lambda x.t_1 t_2)$ повертає $\lambda-$терм $t_1[x:=t_2]$**.** Два стани ядерно еквівалентні, якщо до них застосовне правило редукції. Оперативна функція не визначена на заключних станах. А -процедура – автоматна.

17. Числення Поста. [20,16,1] Числення Поста (відомі ще як канонічні системи Поста) визначається як четвірка $A=(T,V,P,S_0)$, де T - скінченний алфавіт термінальних символів, V - сукупність словарних метазмінних ($T\cap V=\varnothing$), P - скінченна сукупність правил виведення вигляду $u\to v$, $u,v\in(V\cup T)^*, i=\overline{1,n}$, $S_0\subset T^*$ - сукупність аксіом. Говорять, що слово y безпосередньо виводиться зі слова x ($x\Rightarrow_A y$) в системі A, якщо серед його правил існує правило $u=u_0X_1u_1...u_{n-1}X_nu_{n+1}\to\ v=v_0X_{i_1}v_1...v_{m-1}X_{i_m}v_{m+1}$, таке що $x=u_0w_1u_1...u_nw_nu_{n+1}$, $y=v_0w_{i_1}v_1...v_mw_{i_m}v_{m+1}$. Тут підслова u_i утворюють фіксований контекст $u_0X_1u_1...u_{n-1}X_nu_{n+1}$ застосування даного правила. Позначимо $\Rightarrow_A^*$ рефлексивно транзитивне замикання відношення $\Rightarrow_A$. Теорію системи A складають слова, які виводяться з аксіом, тобто $Th(A)=\{y\in T^*: x\Rightarrow_A^* y, x\in S_0\}$.

Мова L породжується системою Поста, якщо існує система Поста A і алфавіт $T'\supset T$ такі, що $L=Th(A)\cap T^*$. Символи з доповнення $T'\setminus T$ є допоміжними в процесі виведення слів мови L. Станами А-процедури системи Поста виступають слова в алфавіті T', заключними станами – слова в алфавіті T. Елементарні правилами виведення відповідають правилам системи і перетворюють слова згідно відношення безпосереднього виведення. Процедура є автоматною з оперативною функцією, яка на стані повертає всі застосовні до нього елементарні правила. Оперативна таблиця процедури є розв'язною, тому процедури ЧП є алгоритмічними.

Алгоритмічні процедурні еквіваленти можуть бути побудовані і для решти існуючих алгоритмічних систем та числень таких, як Таг-системи [20], системи Туе [16], різні підкласи формальних граматики [16], автомати Неймана [25], машини Павлака [26] тощо.

Зазначимо, серед розглянутих моделей алгоритмів разом з А-процедурами, мають ознаку відносності ДП, САА, варіанти ППА, ГС та операторні алгоритми.

Як ми могли впевнитися (див. пп.1-18) :

1) кожній існуючій алгоритмічній моделі притаманний свій тип розв'язності О і свій тип конструктивності в системі процедурних алгоритмів;
2) клас ефективних натуральних функцій замкнений відносно операцій алгебри Чорча. На сьогодні не відомо жодної з таких функції, яка б не належала класу ЧРФ;
3) до класу ЧРФ належать всі натуральні функції, які обчислюються існуючими моделями алгоритмів. Як ми бачили в Розд. 3, не є виключенням і процедурні алгоритми (теорема 3.4);
4) в розглянутих моделях їхня конструктивність спирається на дуже прості елементарні операції штибу тотожній нуль, збільшення числа на 1 та функції проектування (машини Тюрінга, регістрові машини, ЧРФ), приписування символа до слова чи заміну фіксованого підслова на інше фіксоване слово) і примітивну рекурсивність натуральних еквівалентів ОФ.

Свого часу А.Чорч сформулював гіпотезу, яка сьогодні в узагальненому вигляді отримала назву Тези Чорча:

Клас всіх ефективних натуральних функцій збігається з класом ЧРФ, а клас всіх відносно ефективних функцій збігається з класом відносних ЧРФ.

Ця теза отримала в процедурній платформі певне математичне обгрунтування, що випливає з леми 2.1 та теореми 3.4.

Контрольні запитання та вправи

Синтаксичну ознаку ядерної еквівалентності станів ОФ процедури будемо називати коротко – синтаксичною ознакою станів процедури.

1. Що таке регулярна алгебра мов ?

2. Показати, що сукупність всіх: а) номерів мобільної телефонної мережі КиївСтар, б) всіх е- адрес в мережах Google та ukr.net та в) всіх IP-адрес комп'ютерів є регулярними мовами.
3. Що таке скінченний *X*-автомат як А-процедура ? Сформулювати синтаксичну ознаку станів цієї процедури.
4. Побудувати СА для розпізнавання регулярних мов з п.2 та СОТ їхніх А-процедур.
5. Побудувати регулярний вираз і детермінований автомат для: а) множини ідентефікаторівта та б) десяткових констант дійсного типу мови C^{++} та СОТ А-процедур для побудованих автоматів.
6. Довести, що мова $L = \left\{ a^n b^n : n \geq 0 \right\}$ не є регулярною.
7. Що таке скінченний *X*-*Y*-автомат як А-процедура ? Сформулювати синтаксичну ознаку станів цієї процедури.
8. Побудувати *X*-*Y*-автомат для транслітерації текстів з української на англійську мови та СОТ відповідної А-процедури.
9. Що таке МП-автомат як А-процедура ? Сформулювати синтаксичну ознаку станів цієї процедури.
10. Побудувати МП для розпізнавання мови $\left\{ uu^R : u \in \{a,b\}^* \right\}^*$ та СТ відповідної А-процедури.
11. Що таке контекстно-вільні граматики (КВ-граматики) як А-процедури ? Сформулювати синтаксичну ознаку станів цієї процедури.
12. Що таке МП-автомат як А-процедура ? Сформулювати синтаксичну ознаку станів цієї процедури.
13. Мовою Діка в алфавіті $\Sigma_{2n} = \{a_1, b_1, ..., a_n, b_n\}, n > 0$, називається мова D_{2n}, породжена КВ-граматикою $G = (\Sigma_{2n}, N, P, S)$, де $N = \{S\}$, $P = \{S \to SS, S \to \varepsilon, S \to a_i S b_i, S \to a_i S b_i, i = \overline{1,n}\}$. Побудувати МП-автомат для розпізнавання мови Діка D_{2n} та СОТ відповідної А-процедури.
14. Що таке МТ як А-процедура ? Сформулювати синтаксичну ознаку станів цієї процедури.
15. Побудувати МТ для обчислення числової функції $f(x) = 2x - 3$ та СОТ відповідної А-процедури.
16. Що таке НА як А-процедура ? Визначити ядерну еквівалентність ОФ цієї процедури.

17. Побудувати НА для обчислення числової функції $f(x) = 2^{x-1}$ та СОТ відповідної А-процедури.
18. Що таке регістрова машина (РМ) як А-процедура ? Сформулювати синтаксичну ознаку станів цієї процедури.
19. Побудувати РМ для обчислення числової функції $f(x) = 2x - y$ та СОТ відповідної А-процедури.
20. Що таке Дискретні перетворювачі Глушкова (ДП**)** як А-процедура ? Визначити ядерну еквівалентність ОФ цієї процедури.
21. Побудувати ДП для обчислення числової функції НСД $f(x,y) = НСД(x,y)$ за алгоритмом Евкліда через віднімання та СОТ відповідної А-процедури.
22. Що таке Граф-схеми Калужніна (ГС) як А-процедура ? Визначити ядерну еквівалентність ОФ цієї процедури.
23. Побудувати ГС для обчислення числової функції НСД $f(x,y) = НСД(x,y)$ за алгоритмом Евкліда через ділення з операторами присвоєння та СОТ відповідної А-процедури.
24. Довести примітивну рекурсивність натуральних еквівалентів оперативних функцій в алгоритмічних процедурах: а) МТ, б) РМ, в) ГС, г) моделей зі скінченною СОТ і примітивно-рекурсивними синтаксичними ознаками ядерної еквівалентності.
25. У формальних граматиках (ФГ) правила виведення мають вигляд: $u \to w \in P$, $u \in (T \cup N)^*$. Всі решта понять – ті ж, що і в КВ- граматиках. Показати, що як процедури, наступні класи алгоритмічних систем еквівалентні: АМ, МТ, СП та ФГ.

Основна література

1. Нікітченко М. С., Шкільняк С. С. Математична логіка та теорія алгоритмів. Київ: ВПЦ «Київський університет», 2008. 528 с.
2. Зубенко В. В., Омельчук Л.Л. Програмування: навчальний посібник. Київ: ВПЦ «Київський університет», 2011. 625 с.
3. Нікітченко М.С., Шкільняк О.С., Шкільняк С.С. Теорія алгоритмів: навчальний посібник. Київ: Видавничо-поліграфічний центр "Київський університет", 2015. 241 с
4. Кривий С.Л. Дискретна математика. Вибрані питання. Київ: ВД «Києво-Могилянська академія», 2007. 572 с.
5. Моделі обчислень у програмній інженерії. / Глібовець М.М., Кириєнко О.В., Проценко В.С. Київ: Видавничий дім «Києво-Могилянська академія», 2019. 209 с.

Додаткова література

6. Rogers H. Theory of recursive functions end effective computability. McGraw-Hill, Inc., 1967. 482 p.
7. Abramov S.A. Matematichekie postroeniya i programmirovanie. M.: Nauka, 1978. 191 p.
8. Kapitonova Yu.V., Letichevskii A.A. Matematicheskaya teoriya proektirovaniya vichislitelnikh sistem. M.: Nauka, 1988. 295 p.
9. Uspenskii V.A. Cemenov A.L. Teoriya algoritmov: yee osnovnie otkritiya i prilozheniya. V kn. Algoritmi v sovremennoi matematike i yee prilozheniyakh/ pod.red. A.P.Ershova, D. Knuta. N.: VTs SO AN SSSR, 1982. Ch.1. P. 99-342.
10. Kolmogorov A.N., Uspenskii V.A. K opredeleniyu algoritma. *UMN*, T.13, vip.4(82), 1958. P.3-28.
11. Redko V.N., Grishko N.V. Deskriptivnie sistemi: kontseptualnii bazis. *Problemi programuvannya*. N 2-3, 2006. P. 75-80.
12. Glushkov V. M. Abstraktnaya teoriya avtomatov. *UMN*, T.16, vip.5(101), 1961. P. 3-62.
13. Glushkov V. M., Tseitin V. M., Yushchenko Ye. L. Algebra. Yaziki. Programmirovanie. K.: Naukova dumka, 1974. 328 p.
14. Glushkov V.M. Teoriya avtomatov i formalnie preobrazovaniya mikroprogramm. Kibernetika, №5, 1965.
15. Ginzburg S. Mathematical theory of context-free languages. McGraw-Hill, Inc., 1966. 232 p.
16. Gross M., Lanten A. Introduction Formal Grammars. Berlin, Springer-Verlag, 1970. - 231 p.

17. Kleene S.C. Introduction to metamathematics. Van Nostrand Company, Inc. Princeton. 1952. 572 p.
18. Markov A.A. Nagornii N.M. Teoriya algorifmov. M.:Nauka, 1984. 280 p.
19. Cutland Nigel J. Computability. An Introduction to Recursive Function Theory.- Cambridge University Press, 1980. 251 p.
20. Maltsev A. I. Algoritmi i rekursivnie funktsii. M.:Nauka, 1965. 230 p.
21. Kaluzhnin L. A. Ob algoritmizatsii matematicheskikh zadach. V kn. Problemi kibernetiki. M.: Nauka, 1959, Vip.2. P. 51–69.
22. Kotov V.Ye., Sabel'fel'd V.K. Teoriya skhem programm.- M.: Nauka, 1991.- 348 s.
23. Arnold A. Finite Transition System. Paris: Prentice Hall. P. 178 .
24. Barendregt H. The Lambda Calculus: Its Syntax and Semantics. Elsevier Science, 2013. 654 p.
25. Trakhtenbrot B.A. Algorithms and Automatic Computing Machines. HASSELL STREET Press, 2021. 120 p.
26. Pawlak Z. Maszyny programowane. *Algorytmy*, Vol. V, No. 10, 1969. P.7-22.