МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій

(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерної алгебри і дискретної математики

(повна назва кафедри (предметної, циклової комісії))

Дипломна робота

на здобуття ступеня вищої освіти «бакалавр»

(освітньо-кваліфікаційний рівень)

на тему Скінченні автомати

(назва українською)

Finite state machine

(назва англійською)

Виконав: студент заочної форми навчання
спеціальності 123 Комп'ютерна інженерія .

(шифр і назва напряму підготовки, спеціальності)

Мірієв Мірсахіб Мірішевич

(прізвище, ім'я, по-батькові)

Керівник _____

(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент _____

(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:

Протокол засідання кафедри

№___ від «___»_____ 2021 р.

Завідувач кафедри

_____ П.Д. Варбанець

(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК № ___

протокол № __ від «_ __» ____ 2021р.

Оцінка _____/_____/_____

(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____ _____

(підпис) (прізвище, ініціали)

Одеса - 2021

ВСТУП.

У даній дипломній роботі розберемо основні поняття теорії скінченного автомата, а також регулярні вирази, які є частиною сконченых атоматів, розберемо способи задання скінченних автоматів, визначення детермінованого скінченного автомата, недетермінованого скінченного автомата та недетермінованого скінченного автомата з ε переходами. Також напишемо програму, яка буде аналізувати регулярний вираз та текст який ми будем вводити.

Постанова задачі:

1. Розібрати що таке скінчені автомати.
2. Які вони бувають?
3. Що таке регулярні вирази?
4. І для чого регулярні вирази потрібні в скінчених автоматах?

1. Основні теоретичні поняття.

1.1. Основні поняття скінченних автоматів.

Скінченні автомати є моделлю для багатьох компонентів апаратного і програмного забезпечення. Приклади використання скінченних автоматів:

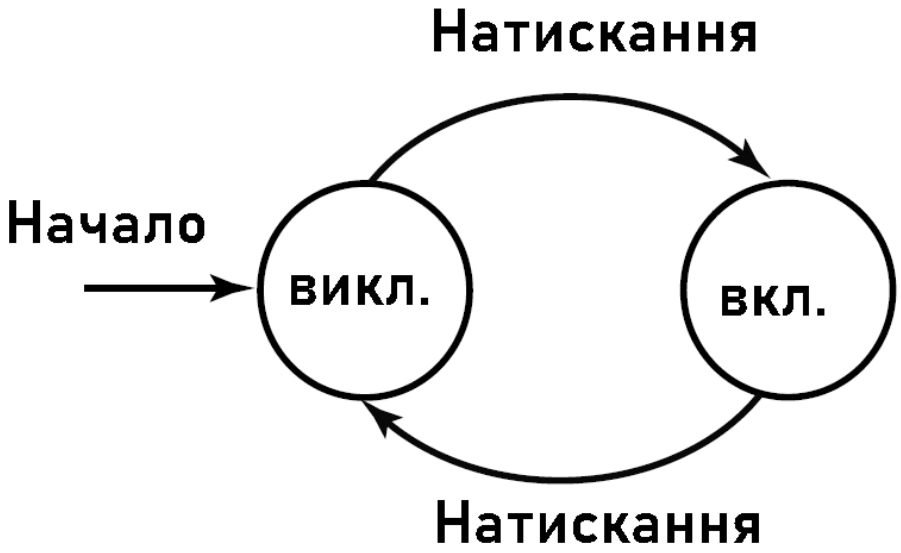
1) Програмне забезпечення використовується для розробки та перевірки цифрових схем.
2) «Лексичний аналізатор» стандартного компілятора, тобто той компонент компілятора, який відповідає за розбивку вихідного тексту на такі логічні одиниці, як ідентифікатори, ключові слова і знаки пунктуації.
3) Програмне забезпечення для сканування таких великих текстових масивів, як набори WEB-сторінок, з метою пошуку заданих слів, фраз або інших послідовностей символів (шаблонів).
4) Програмне забезпечення для перевірки різного роду систем (протоколи зв'язку або протоколи для захищеного обміну інформацією) які можуть знаходитися в кінцевому числі різних станів.

Під словом автомат ми розуміємо метод який дозволяє нам узнавати або розпізнавати елементи з **Σ***.

Так я елементи які розпізнає автомат утворюють множества в **Σ*** , то вони утворюють саму мову. Тому кожен автомат буде узнавати або розпізнвати мову в **Σ***.Мова в **Σ*** складається зі слів розпізнається автоматом .

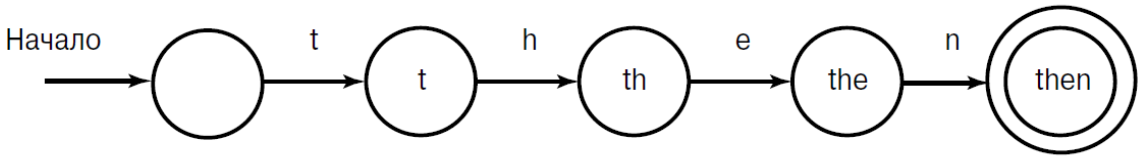
Приклад 1 : Найпростішим нетривіальним скінченним автоматом є перемикач «включено-виключено». Цей пристрій пам'ятає свій поточний стан і від цього стану залежить результат натискання кнопки. Зі стану «вимкнено» натискання кнопки переводить перемикач в стан «включено» і навпаки.

На малюнку 1.1 представлена модель скінченого автомата для перемикача. Як і для всіх скінченних автоматів стан позначено кружком. У цьому випадку це стан «вкл.» та «викл.». Дуги між станами позначені як вхідні символи, які задають зовнішні впливи на систему.



Малюнок 1.1

Приклад 2: Іноді стан автомата запам’ятовую більш складну інформацію ніж «вкл.-викл.». На малюнку 1.2 представлений скінченний автомат який може служити частиною лексичного аналізатора. Його функцією є розпізнавання ключового слова *then*. Цей автомат повинен мати 5 різних станів, кожен з яких представляє позицію слова *then*, досягнуту на даний момент. Ці позиції відповідають префіксам слова, від порожнього ланцюжка до цілого слова.



Малюнок 1.2

1.2. Приклад скінченного автомата у реальному житті.

Розглянемо протокол підтримуючий операції з «електронними грошима»- файлами, які клієнт використовую для оплати за товари в *Internet*, а продавець отримує з гарантією, що гроші -справжні. Для цього продавець повинен знати що ці файли не були підроблені або скопійовані і відіслані продавцеві, хоча клієнт зберігає копію цього файлу і знов використовую її для оплати. Неможливість підробки файлу повинна бути гарантована банком і стратегією шифрування. Таким чином, третій учасник, банк, повинен випускати і шифрувати «грошові» файли так, щоб виключити можливість підробки. Але у банка є і інше важливе завдання: зберігати у своїй базі даних інформацію про всі видані ними гроші. Це потрібно для того, щоб банк міг підтвердити, що отриманий магазином файл являє реальні гроші і може бути переведений на рахунок магазину.

Однак для того щоб використовувати електроні гроші, необхідно скласти протоколи, що дозволяють виробляти різні дії, залежно від бажання користувача. Оскільки в монетарних системах завжди можливе шахрайство, потрібно перевіряти правильність використання грошей, яка б система шифрування не застосовувалась. Іншими словами потрібно гарантувати що статися можуть тільки гарантовані події. Це не дозволить вкрасти гроші. Зараз приведу приклад протоколу:

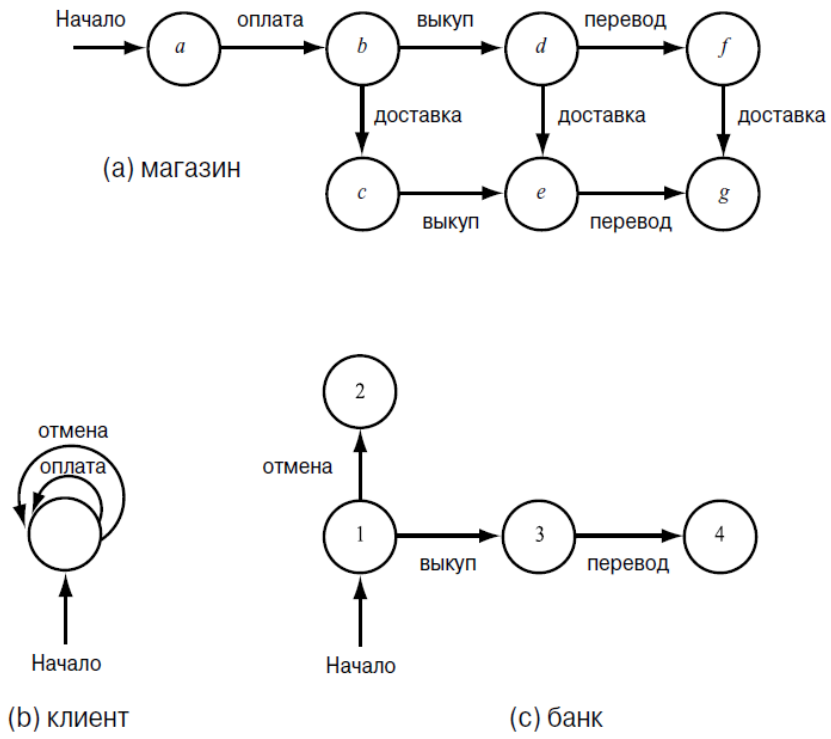
Є три учасники: клієнт, магазин і банк. Для простоти припустимо що є лише один грошовий файл «гроші». Клієнт може прийняти рішення передати його в магазин, який потім обміняє його в банку (точніше потребує щоб банк випустив новий файл, який вже буде належати магазину, а не клієнту, у свою чергу магазин віддають товар клієнту.)

Взаємодія усіх трьох учасників обмежена 5-ю діями:

1) Клієнт може здійснити оплату (*pay*) товару, тобто переслати грошовий файл в магазин.

2) Клієнт може виконати скасування (*cancel*) грошей. Вони відправляються в банк разом з повідомленням про те, що їх суму слід додати до банківського рахунку клієнта.
3) Магазин може призвести доставку (*ship*) товару клієнту.
4) Магазин може зробити викуп (*redeem*) грошей. Вони відправляються в банк разом з вимогою передати їх суму магазину.
5) Банк може виконати переказ (*transfer*) грошей, створивши новий, незалежним чином зашифрований, файл і переславши його магазину.

На малюнку 1.2.1 зображена схема цих дій.

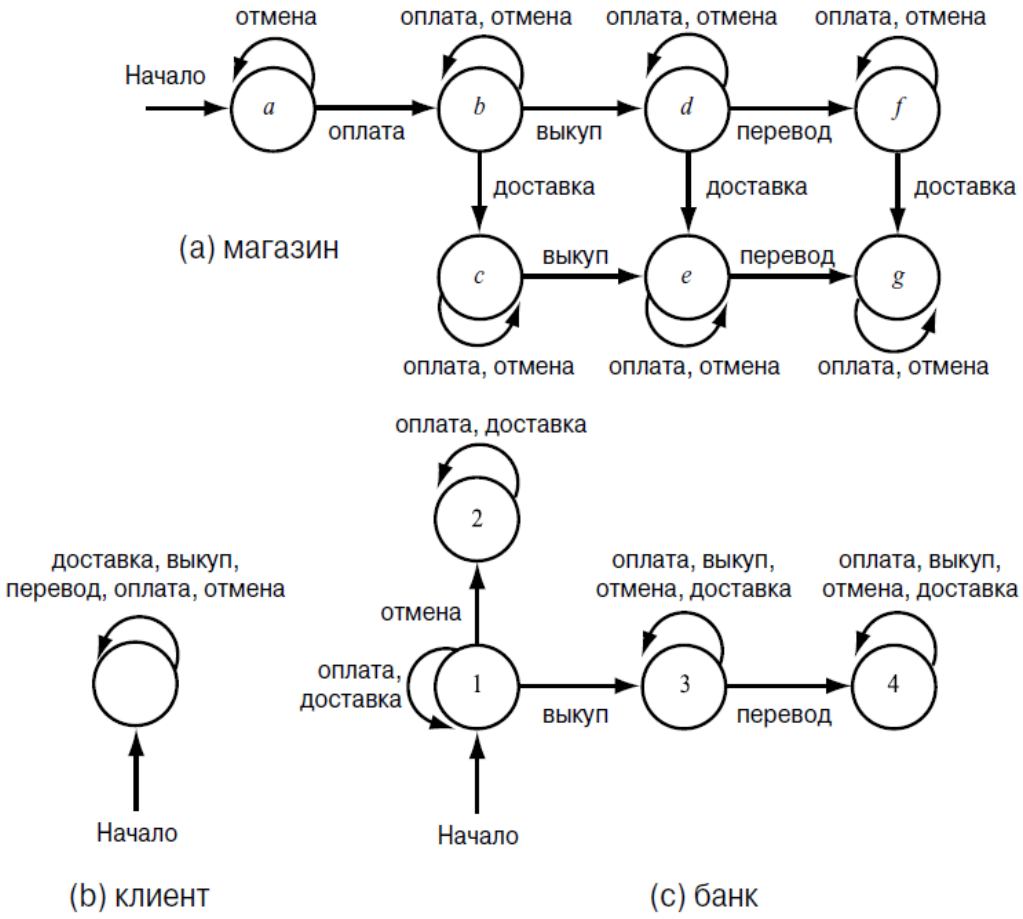


Малюнок 1.3

Потрібно пам'ятати, що тут мова йде про однин грошовий файл. Насправді, банк буде робота за таким же протоколу з великою кількістю

одиниць електронних грошей. При цьому кожен раз протокол роботи з кожною такою одиницею - один і той же, тому ми можемо переглядати дану проблему так, як якщо б існувала всього одна одиниця електронних грошей.

Стан автоматів на малюнку 1.3 потрібно додати петлі з мітками, які позначають діями, які слід проігнорувати в цих станах. Доповнені таким чином автомати зображені на малюнку 1.3. Для економії місця дуги з різними мітками, які мають початок і кінець в одному і тому ж стані, об'єднуються в одну дугу з де кілками мітками. Ігноруватися повинні наступні два типи дій.



Малюнок 1.4

Дії, що не торкаються даного учасника. Як ми бачили, для магазину єдиною такою дією є скасування, тому кожне його стан має петлю з міткою скасування. До банку не мають відношення ні оплата, ні доставка, а тому до кожного з його станів додається петля з мітками оплата і доставка. Клієнта не торкаються доставка, викуп і переклад, і ми додаємо дуги з такими мітками.

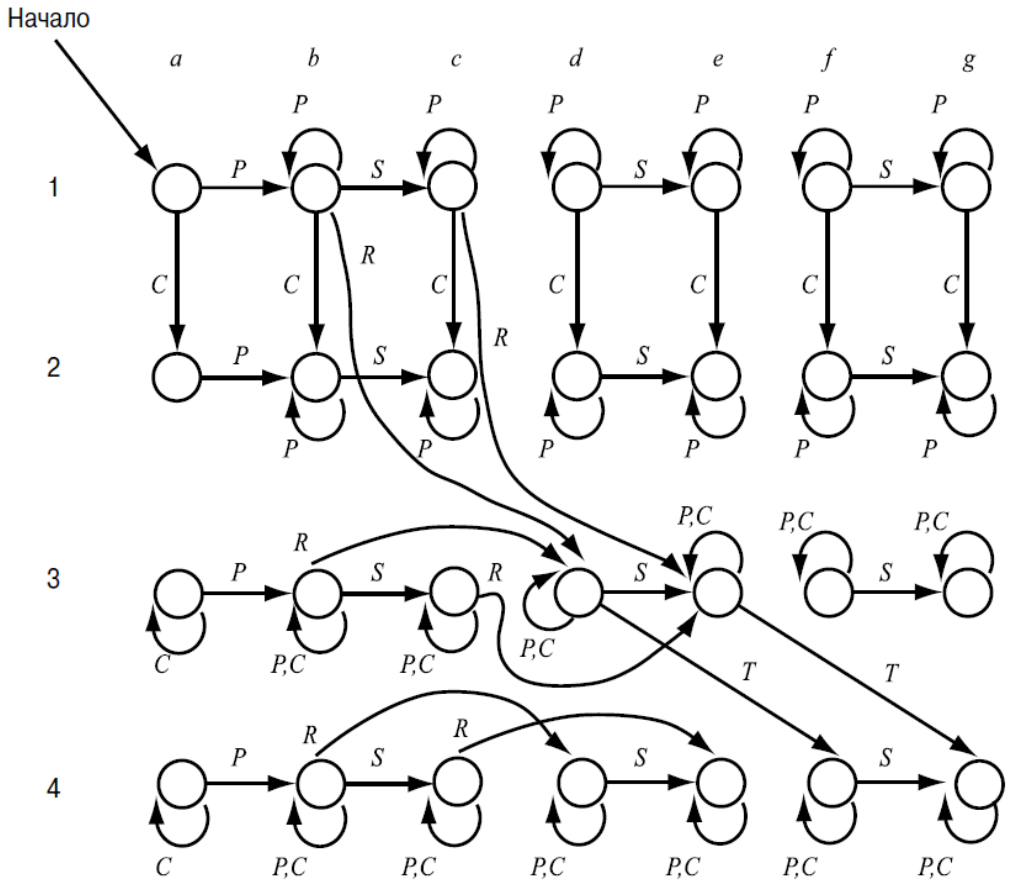
У підсумку, яка б послідовність дій не була подана на вхід, він залишається в своєму єдиному стані. Тому на операції, що здійснюються системою в цілому, автомат клієнта не впливає. Безумовно, клієнт залишається учасником, так як саме він ініціює дії оплата і скасування. Але, як ми вже говорили, для поведінки автоматів неважливо, хто саме ініціює ті чи інші дії.

Дії, які не слід допускати, щоб уникнути смерті автомата. Як згадувалося раніше, щоб не вбити автомат магазину, не можна дозволяти покупцеві повторно виконувати оплату. З цією метою додані петлі з міткою оплата до всім його станів, за винятком стану a (в якому дія оплата доречно і очікувано). Крім того, додані петлі з міткою скасування до станів 3 і 4 банку для того, щоб не допустити смерті автомата банку, якщо покупець спробувати скасувати гроші, які вже були викуплені. Банк з повним правом ігнорує таку вимогу. Точно так же стану 3 і 4 мають петлі з міткою викуп. Магазин не повинен намагатися двічі викупити одні і ті ж гроші. Але якщо він все ж робить це, то банк абсолютно справедливо ігнорує другу вимогу.

1.3. Система в цілому як автомат.

Звичайний спосіб вивчення взаємодії подібних автоматів полягає в побудові так званого автомата-твори. Станами цього автомата є пари стану, перше з яких є стан магазину, а друге - стан банку. наприклад, стан автомата-твори (3, d) являє ситуацію, коли банк перебуває в стані 3, а магазин - в стані d. Оскільки магазин має чотири стану, а банк - сім, то число станів автомата-твори дорівнює $4 \times 7 = 28$.

Цей автомат зображено на малюнку 1.5.



Малюнок 1.5

Тепер ясно, яким чином обрані дуги на малюнку 1.5. Наприклад, отримуючи на вхід дію оплата, магазин здійснює перехід зі стану a в стан b, а з будь-якого іншого стану - в нього ж. Банк же, отримуючи на вхід дію оплата, в будь-якому випадку зберігає свій стан, оскільки ця дія його не стосується. ці спостереження пояснюють, як були побудовані чотири дуги з міткою P в чотирьох стовбцях.

2. Детермінований скінченний автомат.

Детермінований скінченний автомат (ДСА) може перебувати тільки в одному стані. Термін "детермінований" говорить про те, що для будь-якої послідовності вхідних символів існує лише один стан, в яке автомат може перейти з поточного.

ДСА складеться з наступних компонентів:

1) Кінцеве безліч станів, що позначається зазвичай як Q.
2) Кінцеве безліч вхідних символів, що позначається зазвичай як Σ.
3) Функція переходів, аргументами якої є поточний стан і вхідний символ, а значенням новий стан. Функція переходів зазвичай позначається як δ. Представляючи нестрогий автомат у вигляді графа, ми зображували δ зазначеними дугами, що з'єднують стану. Якщо q - стан і a - вхідний символ, то δ (q, a) - це стан p, для якого існує дуга, зазначена символом a і провідна з q в p^2.
4) Початковий стан, один зі станів в Q.
5) Безліч заключних, або допускають, станів F. Безліч F є підмножиною Q.

Перше, що слід з'ясувати про ДСА, - це зрозуміти, яким чином ДСА вирішує, "допускати" чи послідовність вхідних символів. "Мова" ДСА - це безліч всіх його допустимих ланцюжків. Нехай a1a2 ... an - послідовність вхідних символів. ДСА починає роботу в початковому стані q0. Для того щоб визначити стан, в який A перейде після обробки першого символу a1, ми звертаємося до функції переходів δ. Нехай, наприклад, δ (q0, a1) = q1. Для наступного вхідного символу a2 знаходимо δ (q1, a2). Нехай це буде стан q2. Аналогічно знаходяться і наступні стану q3, q4, ..., qn, де δ (qi-1, ai) = qi для кожного i. Якщо qn належить множині F, то вхідна послідовність a1a2 ... an допускається, в іншому випадку вона "відкидається" як неприпустима.

2.1. Просте уявлення про ДСА.

Визначення ДСА як п'ятірки об'єктів з детальним описом функції переходів занадто сухе і не зручне. Існує два більш зручних способу опису автоматів:

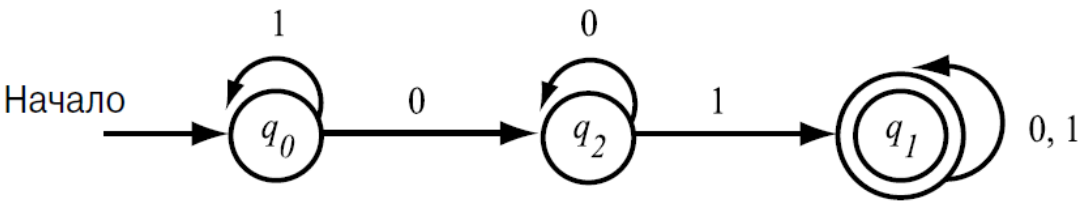
1) Діаграма переходів, яка представляє собою граф
2) Таблиця переходів, що дає табличне представлення функції δ. З неї очевидні
   стану і вхідний алфавіт.

2.2. Діаграма переходів

Діаграма переходів для ДСА виду $A = (Q, \Sigma, \delta, q_0, F)$ є граф, який визначається склад

таким чином:

1) всякому станом з Q відповідає деяка вершина;
2) нехай δ (q, a) = p для деякого стану q з Q і вхідного символу a з Σ. Тоді діаграма переходів повинна містити дугу з вершини q в вершину p, зазначену a. Якщо існує декілька вхідних символів, які переводять автомат зі стану q в стан p, то діаграма переходів може містити одну дугу, зазначену списком цих символів;
3) діаграма містить стрілку в початковий стан, зазначену як начало. Ця стрілка не виходить ні з якого стану;
4) вершини, відповідні допускає станів (станів з F), відзначаються подвійним кружком. Стану, які не належать F, зображені простим (одинарним) кружком.



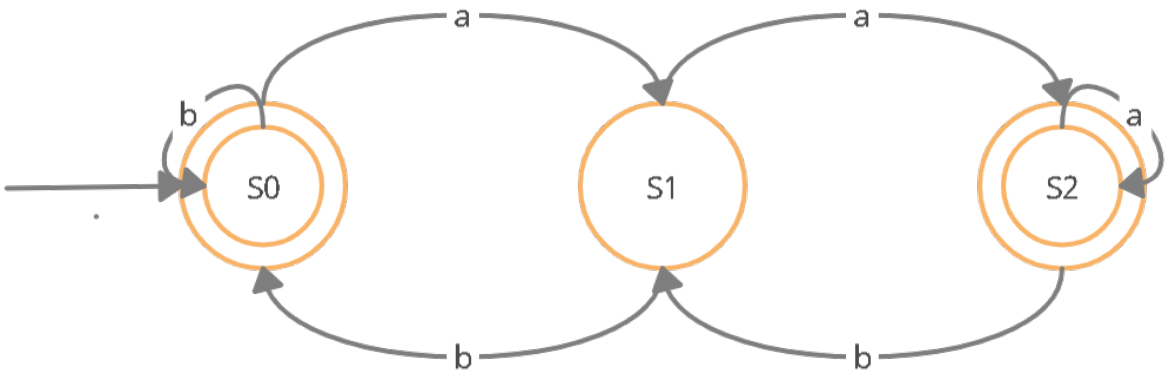
Малюнок 2.1

Приклад: На малюнку 2.1 зображена діаграма переходів для ДСА. На діаграмі видно три вершини, що відповідають трьом станам, стрілка Початок, ведуча в початковий стан q0, і одне допускає стан q1, зазначене подвійним кружком. З кожного стану виходять дві дуги: одна відзначена 0, друга - 1 (для стану q1 дуги об'єднані в одну). Кожна дуга відповідає одному з фактів для δ.

Нехай M – автомат з алфавітом Σ = {a,b}, S = (S0,S1,S2) та δ задан таблицею.

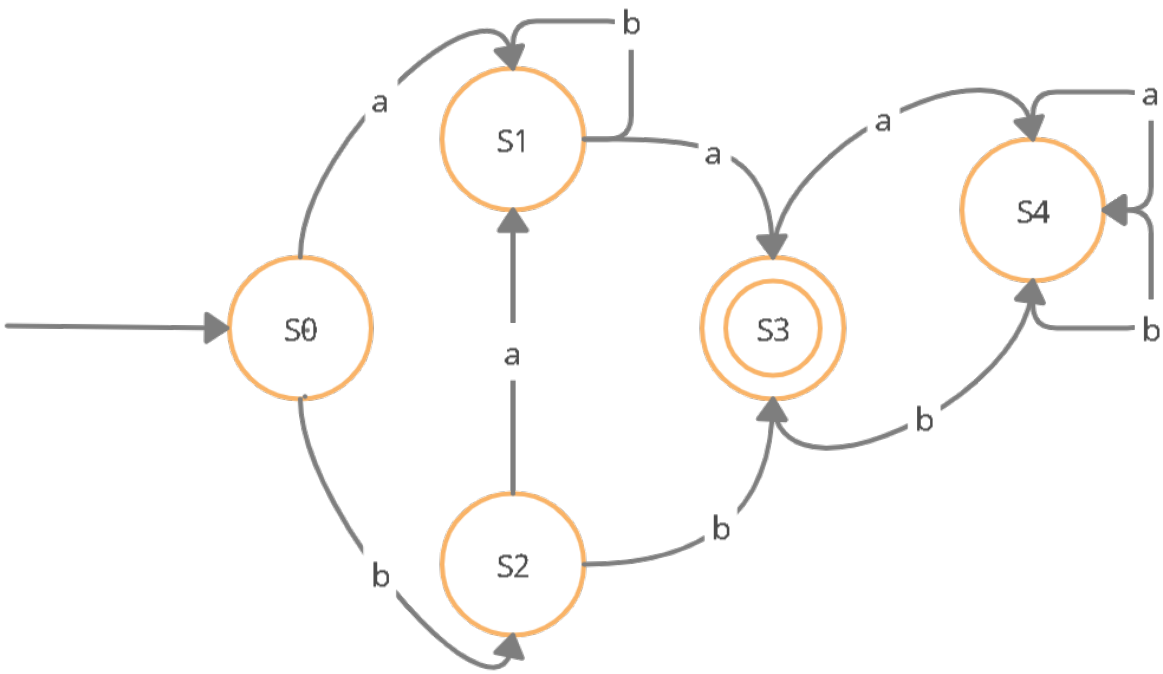
δ	S0	S1	S2
a	S1	S2	S2
b	S0	S0	S1

Малюнок 2.2



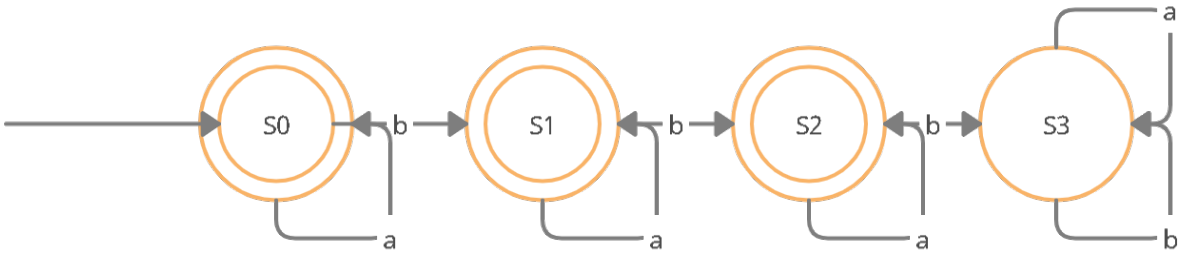
Малюнок 2.3

На малюнку 2.3 зображен скінчений автомат якій задан таблицею (малюнок 2.2). Це детермінований скінчений автомат, де S0 та S2 – стан розпознавання. S0 – початковий стан, який приймає алфавіт (a,b). Якщо ми розглянемо на вхід слово aba, то автомат не розпізнаю його, тому що закінчується у стані S1, який не є станом розпізнавання.



Малюнок 2.4

Автомат на малюнку 2.4 може рознаватися лише в стані S3, тому що тут S3 в цьому автоматі є станом розпізнавання. Що примічательно для цього автомата стан S4 – є станом поглащенія, тому що якщо автомат попадаю в стан S4, то який би не був далі символ автомат все одно буде в стані S4.



Малюнок 2.5

На автоматі малюнок 2.5 всі стані крім стану S3 є станом розпізнавання, а стан S3 – стан поглащенія.

2.3 Таблиці переходів.

Таблиця переходів являє собою звичайне табличне представлення функції, подібної δ, яка двох аргументів ставить у відповідність одне значення. рядки таблиці відповідають станам, а стовпці - вхідним символам. На перетині строки, відповідної станом q, і стовбці, відповідного вхідного символу a, знаходиться стан δ (q, a).

Приклад: На малюнку 2.6 представлена таблиця переходів, відповідна функції δ. Крім того, тут показані і інші особливості таблиці переходів. Начального стану відзначено стрілкою, а допускає - зірочкою. оскільки прописні символи рядків і стовпців задають безлічі станів і символів, у нас є вся інформація, необхідна для однозначного опису даного кінцевого автомата.

	0	1
$\rightarrow q_0$	q_2	q_0
$^*q_1$	q_1	q_1
q_2	q_2	q_1

Малюнок 2.6

3. Недетермінований скінченний автомат.

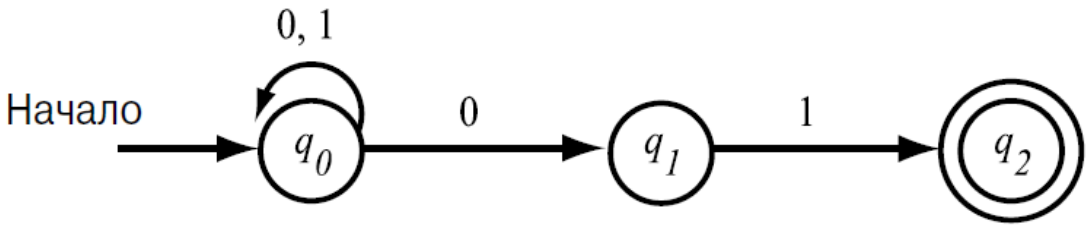
"Недетермінований скінченний автомат, або НСА (NFA - Nondeterministic Finite Automaton), має властивість перебувати в кількох станах одночасно. Цю особливість часто представляють як властивість автомата робити "здогади" щодо його вхідних даних. Так, коли автомат використовується для пошуку певних ланцюжків символів (наприклад, ключові слова) в текстовому рядку великої довжини, то на початку пошуку корисно "здогадатися", що ми знаходимося на початку однієї з цих ланцюжків, а потім використовувати деяку послідовність станів для простої перевірки того, що символ за символом з'являється даний ланцюжок.

Перш, ніж перейти до додатків, потрібно визначити недетерміновані скінчені автомати і показати, що всякий такий автомат допускає мову, яка допустима де кілка ДСА, тобто НСА допускають регулярні мови точно так же, як і ДСА. Однак є причини розглядати і НСА. Вони часто більш компактні і легше будуються, ніж ДСА. Крім того, хоча НСА завжди можна перетворити в ДСА, останній може мати Експоненціально більше станів, ніж НСА, але такі випадки досить рідкісні.

НСА, як і ДСА, мають кінцеве безліч станів, кінцеве безліч вхідних символів, одне початковий стан і безліч допускають станів. Є також функція переходів, яка, як завжди, позначається через δ. Різниця між ДСА і НСА полягає в типі функції δ. У НСА δ є функція, аргументами якої є стан і вхідний символ (як і в ДСА), а значенням - безліч, перебуваючи з нуля, одного або декількох станів (а не один стан, як в ДСА). Перш ніж дати суворе визначення НСА, розглянемо приклад.

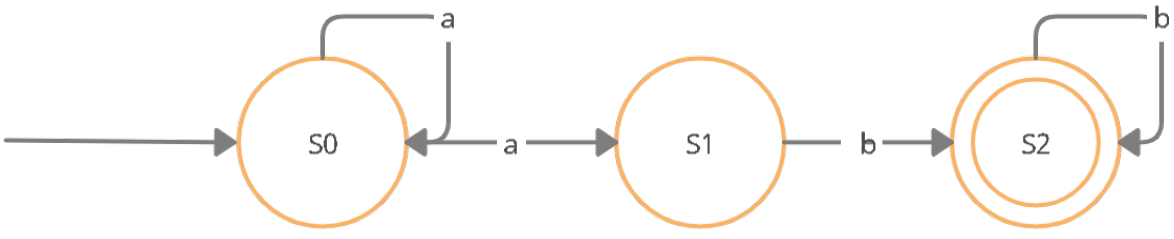
Приклад: На малюнку 3.1 зображений недетермінований скінчений автомат, який допускає ті і тільки ті ланцюжки з 0 і 1, які закінчуються на 01. Початковим є стан q_0, і можна вважати, що автомат знаходиться в цьому стані (а також, можливо, і в інших станах) до тих пір, поки не "здогадається",

що на вході почалась замикаюча підцепочка 01. Завжди існує ймовірність того, що наступний символ не є початковим для замикає підцепочкі 01, навіть якщо це символ 0. Тому стан q_0 може мати переходи в собі як по 1, так і по 0.



Малюнок 3.1

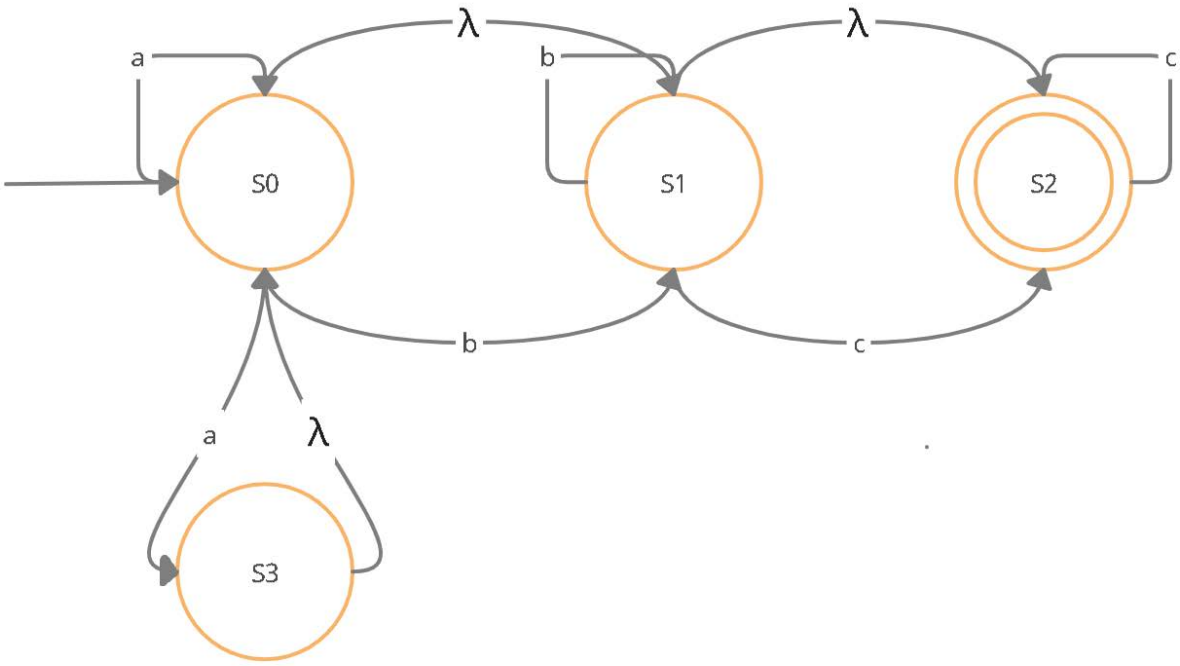
Якщо черговий вхідний символ - 0, то НСА може припустити, що вже почалася замикає підпослідовність 01. Таким чином, дуга з міткою 0 веде зі стану q_0 в q_1. Зауважимо, що з q_0 виходять дві дуги, відмічені символом 0. НСА може перейти як в q_0, так і в q_1, і в дійсності переходить в обидва ці стани. Ми переконаємося в цьому, коли дамо суворе визначення НСА. У стані q1 наш НСА перевіряє, чи є наступний вхідний символ одиницею. Якщо це так, то він переходить в стан q_2 і вважає ланцюжок допустимої.



Малюнок 3.2

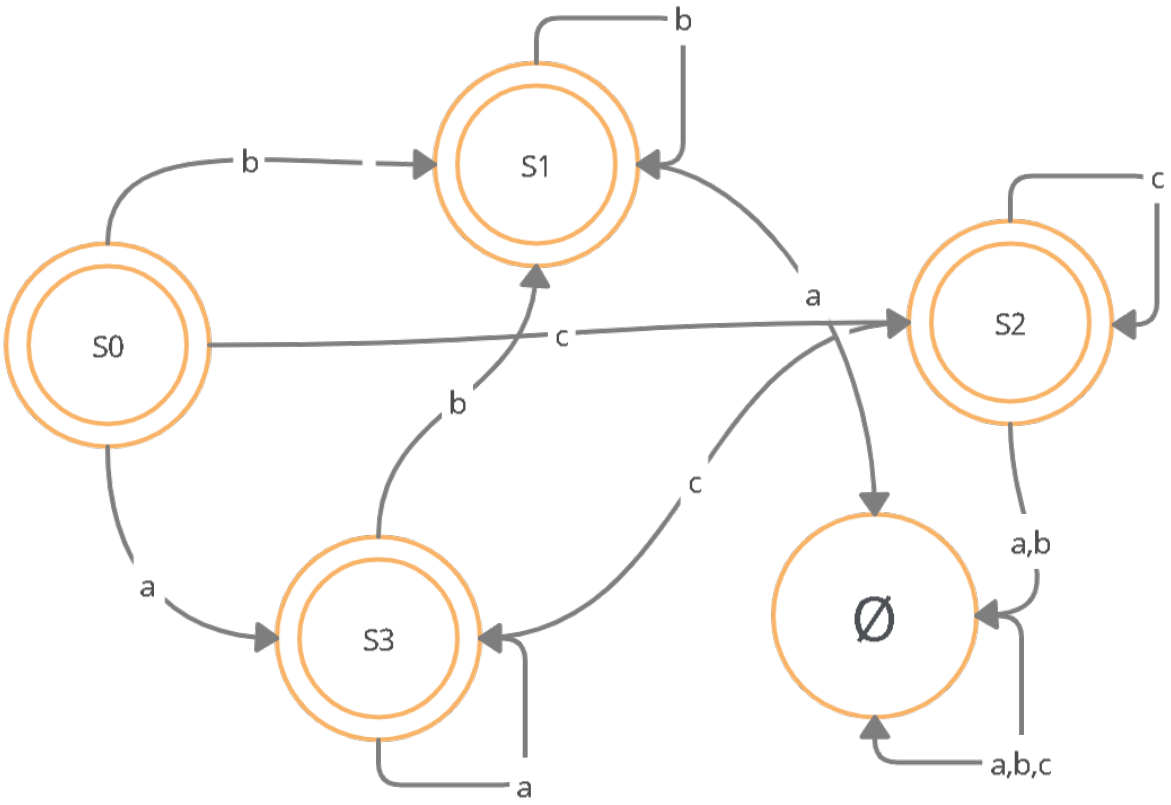
На малюнку 3.2 зображен недетермінованний скінчений автомат, який з початкового стану S0 може вийти лише через символ a, а зі стану S1 тільки

через символ b. S2 – стан розпізнавання. Для цього автомата діє регулярний вираз aa*bb*.



Малюнок 3.3

Автомат на малюнку 3.3 має перехід з S3 в S0 через символ **λ**, тобто пустий символ.



Малюнок 3.4

Автомат на малюнку 3.4 називають «вільним», тобто з початкового стану автомат може перейти з будьякого вхідного символа його алфавіта. Ще в цьому автоматі кожен стан є станом розпізнавання.

3.1 Основні відмінності між ДСА і НСА.

Тепер ми визначимо формально поняття, пов'язані з недетермінованими скінченими автоматами, виділивши по ходу відмінності між ДСА і НСА. Структура НСА в основному повторює структуру ДСА:

$A = (Q, ©, ™, q0, F)$.

Ці позначення мають наступний сенс.

1) Q є кінцеве безліч станів.
2) Σ є кінцеве безліч вхідних символів.
3) q_0, один з елементів Q, - початковий стан.
4) F, підмножина Q, - безліч заключних (або допускають) станів.
5) δ, функція переходів, - це функція, аргументами якої є стан з Q і вхідний символ з Σ, а значенням - деяка підмножина безлічі Q. Помітимо, що єдина відмінність між НСА і ДСА полягає в типі значень функції δ. Для НСА - це безліч станів, а для ДСА - одиночний стан.

Приклад: НСА на малюнку 3.1 можна формально задати, як *({Q0, q1, q2}, {0, 1}, δ, q0, {q2})*, де функція переходів δ задається таблицею на рис. 3.2.

	0	1
$\rightarrow q_0$	$\{q_2, q_1\}$	$\{q_0\}$
q_1	$\varnothing$	$\{q_2\}$
$^*q_2$	$\varnothing$	$\varnothing$

Малюнок 3.5

Зауважимо, що, як і для ДСА, функцію переходів НСА можна задавати таблицею. Єдина відмінність полягає в тому, що в таблиці для НКА на перетинах рядків і стовпців стоять безлічі, хоча, можливо, і одноелементні, тобто містять один елемент (singleton). Зауважимо також, що, коли з деякого стану по певному символу переходу немає, на перетині відповідних рядка та стовпчика має стояти ∅ - порожня множина.

3.2. Розширена функція переходів.

Для НСА, так само, як і для ДСА, нам буде потрібно розширити функцію δ до функції Λδ, аргументами якої є стан q і ланцюжок вхідних символів w, а значенням - безліч станів, в які НСА потрапляє зі стану q, обробивши ланцюжок w.

Базис. (q, ε) = {q}, тобто, не прочитавши жодних вхідних символів НСА знаходиться тільки в тому стані, в якому починав.

Індукція. Припустимо, ланцюжок w має вигляд w = xa, де a - останній символ ланцюжки w, а x - її частина, що залишилася. Крім того, припустимо, що Λδ (q, x) = {p1, p2, ..., pk}.

Нехай:

$$\bigcup_{i=1}^{k} \delta\,(p_i, a) = \{r_1, r_2, \ldots, r_m\}.$$

Тоді: Λδ

(q, w) = {r1, r2, ..., rm}. Говорячи менш формально, для того, щоб знайти Λδ

(q, w), потрібно знайти Λδ (q, x), а потім зробити з усіх отриманих станів усіх переходів по символу a.

Приклад: використовуємо Λδ для опису того, як НСА на малюнку 3.1 обробляє ланцюжок 00101.

1. $\hat{\delta}(q_0, \varepsilon) = \{q_0\}$.
2. $\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0, q_1\}$.
3. $\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \varnothing = \{q_0, q_1\}$.
4. $\hat{\delta}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$.
5. $\hat{\delta}(q_0, 0010) = \delta(q_0, 0) \cup \delta(q_2, 0) = \{q_0, q_1\} \cup \varnothing = \{q_0, q_1\}$.
6. $\hat{\delta}(q_0, 00101) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$.

3.3 Еквівалентність детермінованих і недетермінованих скінчених автоматів.

Для багатьох мов, зокрема, для мови ланцюжків, що закінчуються на 01 побудувати відповідний НСА набагато легше, ніж ДСА. Незважаючи на це, щоб усякий язик, який описується деяким НСА, можна також описати і деяким ДСА. Крім того, цей ДСА має, як правило, приблизно стільки ж станів, скільки і НСА, хоча часто містить більше переходів. Однак в гіршому випадку найменший ДСА може містити 2^n станів, в той час як НСА для того ж самого мови має всього n станів.

У доказі того, що ДСА володіють усіма можливостями НСА, використовується одна важлива конструкція, звана конструкцією підмножин, оскільки включає побудова всіх підмножин множини станів НСА. Взагалі, в доказах тверджень про автоматах часто по одному автомату будується інший. Для нас конструкція підмножин важлива як приклад того, як один автомат описується в термінах станів і переходів іншого автомата без знання специфіки останнього.

Побудова підмножин починається, виходячи з НСА N = (QN, Σ, δN, q0, FN). Метою є опис ДСА D = (QD, Σ, δ_D, {q0}, FD), у якого L (N) = L (D). Відмітимо, що вхідні алфавіти цих двох автоматів збігаються, а початковий стан D є множиною, що містить тільки початковий стан N. Інші компоненти D будуються наступним чином.

• Q_D є безліч всіх підмножин Q_N, або булеан безлічі Q_N. Відзначимо, що якщо Q_N містить n станів, то Q_D буде містити вже 2^n станів. однак часто не всі вони досяжні з початкового стану автомата D. Такі недосяжні стану можна "відкинути", тому фактично число станів D може бути набагато менше, ніж 2^n.

• F_D є безліч підмножин S безлічі Q_N, для яких S ∣ $F_N \neq \emptyset$, тобто F_D складається з усіх множин станів N, що містять хоча б одне допускає стан N.

•Для кожної множини$S \subseteq Q$N і кожного вхідного символа *a* из Σ

$$\delta_D(S, a) = \bigcup_{p \text{ из } S} \delta_N(p, a).$$

Таким чином, для того, щоб знайти δ_D (S, a), ми розглядаємо всі стани p з S, шукаємо ті стани N, в які можна потрапити зі стану p по символу a, а потім беремо об'єднання множин знайдених станів за всіма станів p.

Приклад: Нехай N - автомат на малюнку 3.1, що допускає ланцюжки, які закінчуються на 01. Оскільки безліч станів N є $\{q_0, q_1, q_2\}$, то конструкція підмножин дає ДСА з 23 = 8 станами, такими, що відповідають всім підмножини, складений з цих трьох станів. На малюнку 3.6 наведена таблиця переходів для отриманийних восьми станів. Пояснимо коротко, як були отримані елементи цієї таблиці.

	0	1
$\varnothing$	$\varnothing$	$\varnothing$
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	$\varnothing$	$\{q_2\}$
$^*\{q_2\}$	$\varnothing$	$\varnothing$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$^*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$^*\{q_1, q_2\}$	$\varnothing$	$\{q_2\}$
$^*\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

Малюнок 3.6

Зауважимо, що дана таблиця, елементами якої є безлічі, відповідає детермінованому кінцевому автомату, оскільки стану побудованого ДСА самі є множинами. Для ясності можна переобозначити стан. Наприклад, ∅ позначити як A, $\{q_0\}$ - як B і т.д. Таблиця переходів для ДСА на рис. 3.4 визначає в точності той же автомат, що і на рис. 3.3, і з її вигляду зрозуміло, що елементами таблиці є поодинокі стану ДСА.

	0	1
A	A	A
$\rightarrow B$	E	B
C	A	D
$^*D$	A	A
E	E	F
$^*F$	E	B
$^*G$	A	D
$^*H$	E	F

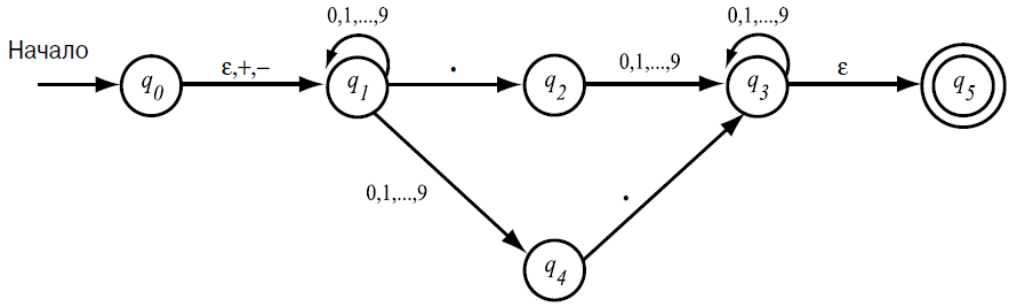
Малюнок 3.7

4. Скінченний автомат з епсилон-переходами.

Розглянемо ще одне узагальнення поняття кінцевого автомата. Додамо автомату нову "властивість" - можливість здійснювати переходи по ε, порожньому ланцюжку, тобто спонтанно, не отримуючи на вхід ніякого символу. Ця нова можливість, як і недетермінізм, не має права продовжувати класу мов, допустимих кінцевими автоматами, але дає деякий додатковий "зручність програмування". Є ще регулярні вирази які тісно зв’язані з НСА, які мають епсилон-переходи. Такі автомати будемо називати ε-НСА. Вони виявляються корисними при доведенні еквівалентності між класами мов, задаються кінцевими автоматами і регулярними виразами.

Приклад: на малюнку 4.1 зображений ε-НСА, що допускає десяткові числа, які складаються з наступних елементів:

1. Необов’язковий знак + чи –.
2. Ланцюг цифр.
3. Розділяє десяткова крапка.
4. Ще один ланцюг цифр. Цей ланцюжок, як і ланцюжок (2), може бути порожнім, але хоча б один з них не порожнім.



Малюнок 4.1

Особливого інтересу заслуговує перехід зі стану q_0 в q_1 за допомогою одного з символів +, - або ε. Стан q_1, таким чином, являє ситуацію, коли прочитаний знак числа, якщо він є, але не прочитана жодна з цифр, ні десяткова крапка. Стан q_2 відповідає ситуації, коли тільки що прочитана

десяткова точка, а цифри цілої частини числа або вже були прочитані, або ні. У стані q_4 вже напевно прочитана хоча б одна цифра, але ще не прочитана десяткова крапка. Таким чином, q_3 інтерпретується як ситуація, коли ми прочитали десяткову точку і хоча б одну цифру зліва або праворуч від неї. Ми можемо залишатися в стані q_3, продовжуючи читати цифри, але можемо і "здогадатися", що ланцюжок цифр закінчена, і спонтанно перейти в допускає стан q_5.

4.1 Формальна запис ε-НСА.

ε-НСА можна представляти так само, як і НСА, з тією лише різницею, що функція переходів повинна містити інформацію про переходах по ε. Формально, ε-НСА *A* можна представити у вигляді $A = (Q, \Sigma, \delta, q0, F)$, де всі компоненти мають таке ж значення, що і для НСА, за винятком δ, аргументами якої тепер є стан з Q і елемент безлічі $\Sigma \cup \{\varepsilon\}$, тобто або деякий вхідний символ, або ε. ніяких непорозумінь при цьому не виникає, оскільки ми обговорюємо, що ε, символ порожній ланцюжка, не є елементом алфавіту Σ.

Приклад: ε-НСА на малюнку 4.1 можна формально представити як

$E = (\{q0, q1, ..., q5\}, \{., +, -, 0, 1, ..., 9\}, \delta, q0, \{q5\})$, де функція переходів δ визначена таблицею переходів на малюнку 4.2.

	ε	+, –	.	0, 1, …, 9
q_0	$\{q_1\}$	$\{q_1\}$	$\varnothing$	$\varnothing$
q_1	$\varnothing$	$\varnothing$	$\{q_2\}$	$\{q_1, q_4\}$
q_2	$\varnothing$	$\varnothing$	$\varnothing$	$\{q_3\}$
q_3	$\{q_5\}$	$\varnothing$	$\varnothing$	$\{q_3\}$
q_4	$\varnothing$	$\varnothing$	$\{q_3\}$	$\varnothing$
q_5	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$

Малюнок 4.2

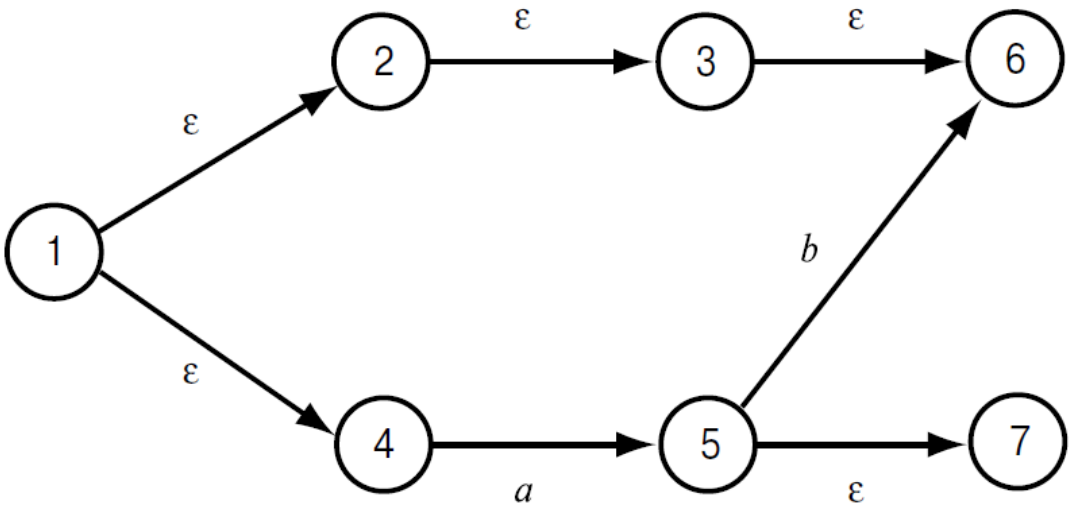
4.2 Що таке ε-замикання.

Дамо формальне визначення розширеної функції переходів для ε-НСА, яка призведе до визначення допустимості ланцюжків і мов для даного типу автоматів і врешті-решт допоможе зрозуміти, чому ДСА можуть імітувати роботу ε-НСА. Але перш потрібно визначити одне з центральних понять, так званого ε-замикання стану. Говорячи нестрого, ми отримуємо ε-замикання стану q, здійснюючи всі можливі переходи з цього стану, відмічені ε. Але після скоєння цих переходів і отримання нових станів знову виконуються ε-переходи, вже з нових стану, і т.д. Зрештою, ми знаходимо все стану, в які можна потрапити з q по будь-якому шляху, кожен перехід в якому відзначений символом ε. Формально ми визначаемо ε-замикання, ECLOSE, рекурсивно наступним чином.

Базис. ECLOSE (q) містить стан q.

Індукція. Якщо ECLOSE (q) містить стан p, і існує перехід, відмічений ε, зі стану p в стан r, то ECLOSE (q) містить r. Точніше, якщо δ є функція переходів розглянутого ε-НСА і ECLOSE (q) містить p, то ECLOSE (q) містить також їхні капітали з $\delta\ (p,\ \varepsilon)$.

Приклад: У автомата, зображеного на малюнку 4.1, кожний стан є власним ε-замиканням, за винятком того, що ECLOSE $(q_0) = \{q_0, q_1\}$ і ECLOSE $(q_3) = \{q_3, q_5\}$. Це пояснюється тим, що є лише два ε-переходу, один з яких додає q_1 в ECLOSE (q_0), а інший - q_5 в ECLOSE (q_3). На малюнку 4.3 наведено більш складний приклад. Для цього в ньому набору станів, який може бути частиною деякого ε-НСА, ми можемо зробити висновок, що ECLOSE (1) = {1, 2, 3, 4, 6}.



Малюнок 4.3

У кожне з цих станів можна потрапити зі стану 1, слідуючи по шляху, відміченому виключно ε. Наприклад, в стан 6 можна потрапити по дорозі 1 → 2 → 3 → 6. Стан 7 не належить ECLOSE (1), оскільки, хоча в нього і можна потрапити з стану 1, у відповідному шляху міститься перехід 4 → 5, зазначений НЕ ε. І не має значення, що в стан 6 можна потрапити зі стану 1, слідуючи також по шляху 1 → 4 → 5 → 6, в якому присутні не ε-перехід. Існування одного шляху, відміченого тільки ε, вже досить для того, щоб стан 6 містилося в ECLOSE (1).

5. Регулярні вирази.

По теоремі Клейна ми знаємо, що алфавіт є регулярним виразом тоді і тільки тоді, коли він допускається скінченним автоматом.

1) Нехай Σ – алфавіт з скінченним множеством символов. Строкою або символом називають послідовність a1, a2, a3 … an, ai є Σ. Тому Σ = {a,b}, то ab, aaab, abbbaba – слова. До цього можно додати пусте слово символом λ.

2) Нехай Σ* це множина усіх кінцевих слів в алфавіті разом з пустим словом. Конкатенація Σ* буде виглядати так:

   a1a2a3 · b1b2b3b4 = a1a2a3b1b2b3b4

Перейдемо від "машинного" завдання мов за допомогою ДСА і НСА до алгебраічному опису мов за допомогою регулярних виразів. Встановимо, що регулярні вираження визначають точно ті ж мови, що і різні типи автоматів, а саме, регулярні мови. У той же час, на відміну від автоматів, регулярні вирази позволяють визначати допустимі ланцюжки декларативним способом. Тому регулярні вирази використовуються в якості вхідної мови в багатьох системах, обробних ланцюжків. Розглянемо два приклади:

1) Команди пошуку, наприклад, команда *grep* операційної системи UNIX або аналогічної команди для пошуку ланцюжків, які можна зустріти в Web-браузерах або системах форматування тексту. У таких системах регулярні вирази використовується для опису шаблонів, які користувач шукає в файлі. Різні пошукові системи перетворять регулярний вираз або в ДСА, або в НСА і застосовують цей автомат до файлу, в якому проводиться пошук.
2) Генератори лексичних аналізаторів, такі як Lex або Flex. Нагадаємо, що лексичний аналізатор - це компонент компілятора, який розбиває вихідну програму на логічні одиниці (лексеми), які складаються з одного або декількох символів і мають певний сенс. Прикладами

лексем є ключові слова (наприклад *while*), ідентифікатори (будь-яка буква, за якою слід нуль або кілька букв і / або цифр) і такі знаки, як + або <=. Генератор лексичних аналізаторів отримує формальні описи лексем, які є по суті регулярними виразами, і створює ДСА, який розпізнає, яка з лексем з'являється на його вході.

Якщо B є **Σ***, то B* це множество слів із B разом з пустим словом.

Якщо **Ø** – пусте слово, то **Ø** = {**λ**}.

Символом * називаємо зіркою Клейна.

Давайте розглянемо це на прикладі.

Наприклад:

{a}* = { λ,a,aa,aaa….};

{a}{ab}*{c} = {ac,aabc,aabababc,….}

Нехай **Σ** – алфавіт. Клас регулярних виразів над **Σ** визначається наступними правилами використання **Σ** і символів **Ø, λ, *, ∪, and**.

1) Символ **Ø** є регулярним виразом та для кожного a є **Σ**, а теж буде регулярним виразом.
2) Якщо w1 та w2 – два регулярних вирази, то w1w2, w1 ∪ w2, w* також регулярні вирази.
3) Нема регулярних виразів які не породжені.

Образ регулярного виразу це і є регулярна мова.

Визначення:

Клас R регулярних мов над **Σ** має наступні властивості.

1) Пусте множество **Ø** є R та якщо a є **Σ**, то {a} є R.

2) Якщо S1 та S2 є R, тоді S1 U S2, S1 · S2 є R.

Приклади регулярних виразів:

(a ∪ b)*, (a* ∪ b*), a* (c ∪ d)a(b ∪ a ∪ c)*.

Приклади регулярних множеств:

{a,b,c} {a}*, {ab}*, {c}{d}* і т.д.

Розберемо приклади вправ для знаходження регулярного множества та регулярного виразу:

Тут ми знаходемо регулярне множество з виразу.

1) a(b ∪ c ∪ d)a = { aba, aca, ada}

2) a*b*c = {c, ac, abc, bc, aac, …}

3) (a ∪ b)(c ∪ d) = {ac, ad, bc, bd}

А тут навпаки, знаходимо регулярний вираз з регулярного множества:

1) {ab, abb, abbb, …} = abb*

2){abcd, abef, cdcd, cdef} = (ab ∪ cd)(cd ∪ ef)

Зробимо висновок і підсумуємо що:

1) Пусте множество **Ø** є регулярною мовою

2) Множество, що складається лише з одної пустої строки теж є регулярною мовою.

3) Множество, що складається з однобуквеного слова ({a}, де a є **Σ**) теж є регулярною мовою.

4) Якщо α і β – регулярні мови, то їх обьеднання, конкатенація та

якщо вони з зіркою Клайна вони теж є регулярною мовою.

5) Інших регулярних мов немає.

5.1 Оператори регулярних виразів.

Регулярні вирази позначають (задають, або представляють) мови. В якості простого прикладу розглянемо регулярний вираз 01 * + 10 *. Воно визначає мову всіх ланцюжків, які перебувають або з одного нуля, за яким слід будь-яку кількість одиниць, або з однієї одиниці, за якою слідує довільну кількість нулів. На даний момент ми не розраховуємо, що читач знає, як інтерпретуються регулярні вирази, тому наше твердження про мову, який задається цим виразом, поки має бути прийнято на віру. Щоб зрозуміти, чому наша інтерпретація заданого регулярного виразу правильна, необхідно визначити всі використані в цьому виразі символи, тому спочатку познайомимося з наступними трьома операціями над мовами, відповідними операторам регулярних виразів.

1. **Об'єднання** двох мов L і M, що позначається L U M, - це безліч ланцюжків, які містяться або в L, або в M, або в обох мовах. Наприклад, якщо L = {001,10, 111} і M = {ε, 001}, то L U M = {ε, 10, 001, 111}.
2. **Конкатенація** мов L і M - це безліч ланцюжків, які можна утворити шляхом дописування до будь-якої ланцюжку з L будь ланцюжка з M. Конкатенація мов позначається або точкою, або взагалі ніяк не позначається, хоча оператор конкатенації часто називають "Точкою". Наприклад, якщо L = {001, 10, 111} і M = {ε, 001}, то L.M, або просто LM, - це {001, 10, 111, 001001, 10001, 111001}. Перші три ланцюжки в LM – це ланцюжка з L, з'єднані з ε. Оскільки ε є одиницею (нейтральним елементом) для операції конкатенації, результуючі ланцюжка будуть такими ж, як і ланцюжки з L. Останні ж три ланцюжки в LM утворені шляхом з'єднання кожного ланцюжка з L з другої ланцюжком з M, тобто з 001. Наприклад, 10 з L, з’єднання з 001 з M, дає 10001 для LM.

3. **Ітерація** ("зірочка", або замикання Кліні2) мови L позначається L * і представляє собою безліч всіх тих ланцюжків, які можна утворити шляхом конкатенації будь-якої кількості ланцюжків з L. При цьому допускаються повторення, тобто одна і той же ланцюжок з L може бути обрана для конкатенації більше одного разу. наприклад, якщо L = {0, 1}, то L * - це все ланцюжки, що складаються з нулів і одиниць. Якщо L = {0, 11}, то в L * входять ланцюжка з нулів і одиниць, що містять парне кількість одиниць, наприклад, ланцюжки 011, 11110 або ε, і не входять ланцюжка 01011 або 101. Більш формально мову L * можна уявити як нескінченне об'єднання $U_{i\geq 0}$ L^i, де $L^0 = \{\varepsilon\}$, $L^1 = L$ і L^i для $i > 1$ дорівнює LL ... L (конкатенація і копій L).

5.2 Побудова регулярних виразів.

Все алгебри починаються з деяких елементарних виразів. Зазвичай це константи і / або змінні. Застосовуючи певний набір операторів до цих елементарних виразів і вже побудованим виразам, можна конструювати більш складні вирази. Зазвичай необхідно також мати деякі методи групування операторів і операндів, наприклад, за допомогою дужок. Наприклад, звичайна арифметична алгебра починається з констант (цілі і дійсні числа) і змінних і дозволяє нам будувати більш складні вирази за допомогою таких арифметичних операторів, як + або ×.

Базис. Базис складеться з трьох частин.

1. Константи ε і ∅ є регулярними виразами, що визначають мови {ε} і ∅, відповідно, тобто L (ε) = {ε} і L (∅) = ∅.
2. Якщо a - довільний символ, то a - регулярний вираз, що визначає мову {A}, тобто L (a) = {a}. Зауважимо, що для запису виразу, відповідного символу, використовується жирний шрифт. Це відповідність, тобто A відноситься до a, має бути очевидним.

3. Змінна, позначена прописаний курсивною буквою, наприклад, L, являє довільний мову.

Індукція. Індуктивний крок складається з чотирьох частин, по одній для трьох операторів і для введення дужок.

1. Якщо E і F - регулярні вирази, то E + F - регулярний вираз, який визнає об'єднання мов L (E) і L (F), тобто L (E + F) = L (E) U L (F).
2. Якщо E і F - регулярні вирази, то EF - регулярний вираз, що визначають конкатенацію мов L (E) і L (F). Таким чином, L (EF) = L (E) L (F). Зауважимо, що для позначення оператора конкатенації - як операції над мовами, так і оператора в регулярному виразі - можна використовувати точку. Наприклад, регулярний вираз 0.1 означає те ж, що і 01, і представляє мову {01}. Однак ми уникаємо використовувати точку в якості оператора конкатенації в регулярних виразах.
3. Якщо E - регулярний вираз, то E * - регулярний вираз, що визначає ітерацію мови L (E). Таким чином, L (E *) = (L (E)) *.
4. Якщо E - регулярний вираз, то (E) - регулярний вираз, що визначає ту саму мову L (E), що і вираз E. Формально, L ((E)) = L (E).

6. Програма.

6.1. Програма аналізатор регулярних виразів.

Код програми.

Програма аналізує регулярний вираз і текст до виразу и знаходить сходження.

```
namespace core;


use interfaces\IAuthenticated;


class API {


    protected $db;

    protected $resultShouldReturn;


    public function execute($values, $returnResult =
false) {

        date_default_timezone_set('UTC');


        ob_start('ob_gzhandler');

        $this->resultShouldReturn = $returnResult;

        $result = null;


        $action = idx($values, 'action');
```

```
        if (empty($action)) {
            $this->result(new
\core\APIError(\core\ErrorCodes::NO_ACTION));
        } else {
            safeRequireOnce("actions/$action.php",
false);

            $action =
formatActionNameForExecution($action);

            if (!is_null($action)) {
                // Any call here could throw.
                try {
                    $action = new $action($values);
                    if ($action->requiresDatabase()) {
                        $this->connect();
                        $action->setDB($this->db);
                    }
                    $result = $action->validate()-
>execute();
                } catch (\core\APIError $err) { //
Custom error
                    $result = $err;
```

```
                } catch (\Exception $err) { // A
unknown PHP error happened
                    $message = array('error' => $err);

                    if (DEBUG) {
                        $message['stack'] =
debug_backtrace(true);
                    }

                    if ($this->db->inTransaction()) {
                        $this->db->rollback();
                    }

                    $result = new
\core\APIError(\core\ErrorCodes::UNKNOWN, $message);
                }

                // Assume success if the action returns
no result. (Ex; a DB write);
                if (is_null($result)) {
                    $result = new \core\Result();
                }
```

```
                $totalTime =
number_format((\microtime(true) -
$_SERVER["REQUEST_TIME_FLOAT"])*1000, 2, '.', '');


                return $this->result($result,
$totalTime.'ms');
            } else {
                return $this->result(new
\core\APIError(\core\ErrorCodes::NO_ACTION, "Action
$action does not exist."));
            }
        }
    }

    function getDB() {
        return $this->db;
    }


    function connect() {
        $this->db = new \core\DB();
        $this->db->connect(DB_HOST, DB_USER_NAME,
DB_PASSWORD, DB_NAME, DB_PORT, DB_SOCK);
    }
```

```
function result($data, $time=null) {
    $success = $data instanceof \core\Result;
    $resultData = array('success' => $success);
    $returnData = $data->getData();
    $metadata = $data->getMetadata();

    if ($success == false) {
        // http_response_code(500);
    }

    if (!is_null($returnData)) {
        $resultData['data'] = $returnData;
    }

    if (!is_null($time)) {
        if (is_null($metadata)) {
            $metadata = array();
        }
        $metadata['script-time'] = $time;
    }

    if (!is_null($metadata)) {
        $resultData['metadata'] = $metadata;
```

```
        }

        echo(json_encode($resultData));

        if ($this->resultShouldReturn) {
            return ob_get_contents();
        }
    }
}
namespace core;

class Result {

    private $data;
    private $metadata;

    function __construct($data = null, $metadata =
null) {
        $this->data = $data;
        $this->metadata = $metadata;
    }

    public function getData() {
```

```
        return $this->data;
    }

    public function getMetadata() {
        return $this->metadata;
    }
}
namespace core;

class Session {

    protected $db;

    public function __construct($db, $name) {
        $this->db = $db;

        session_set_save_handler(
            array($this, "open"),
            array($this, "close"),
            array($this, "read"),
            array($this, "write"),
            array($this, "destroy"),
            array($this, "gc")
```

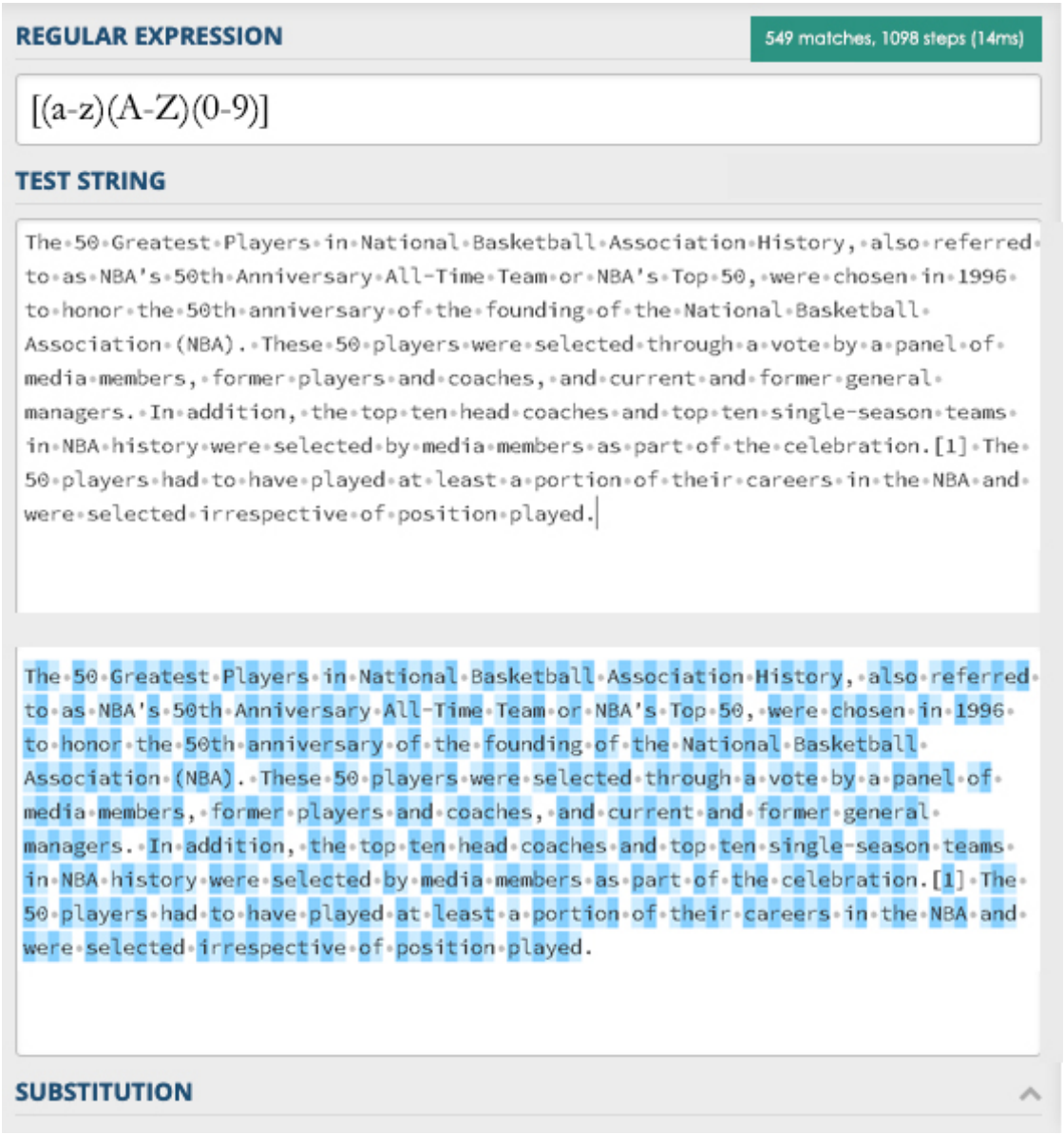
```
        );

        session_name($name);

        if(!isset($_SESSION)) {
            session_start();
        }
    }

    public function open() {
        return $this->db->isConnected();
    }

    public function close() {
        return true;
    }

    public function read($id) {
        try {
            $result = $this->db->execute("SELECT data
FROM sessions WHERE id = ?", ["s", $id], true);
        } catch (\Exception $ex) {
            return '';
```

```
        }
        return !is_null($result)?$result->data:'';
    }

    public function write($id, $data) {
        $lastAccess = time();

        try {
            $sql = "INSERT INTO sessions (id, access,
data)
                VALUES (?, ?, ?)
                ON DUPLICATE KEY UPDATE `access`=?,
`data`=?
            ";

            $result = $this->db->execute($sql, [
                ["s", $id],
                ["s", $lastAccess],
                ["s", $data],
                ["s", $lastAccess],
                ["s", $data],
            ]);
        } catch(\Exception $ex) {
```

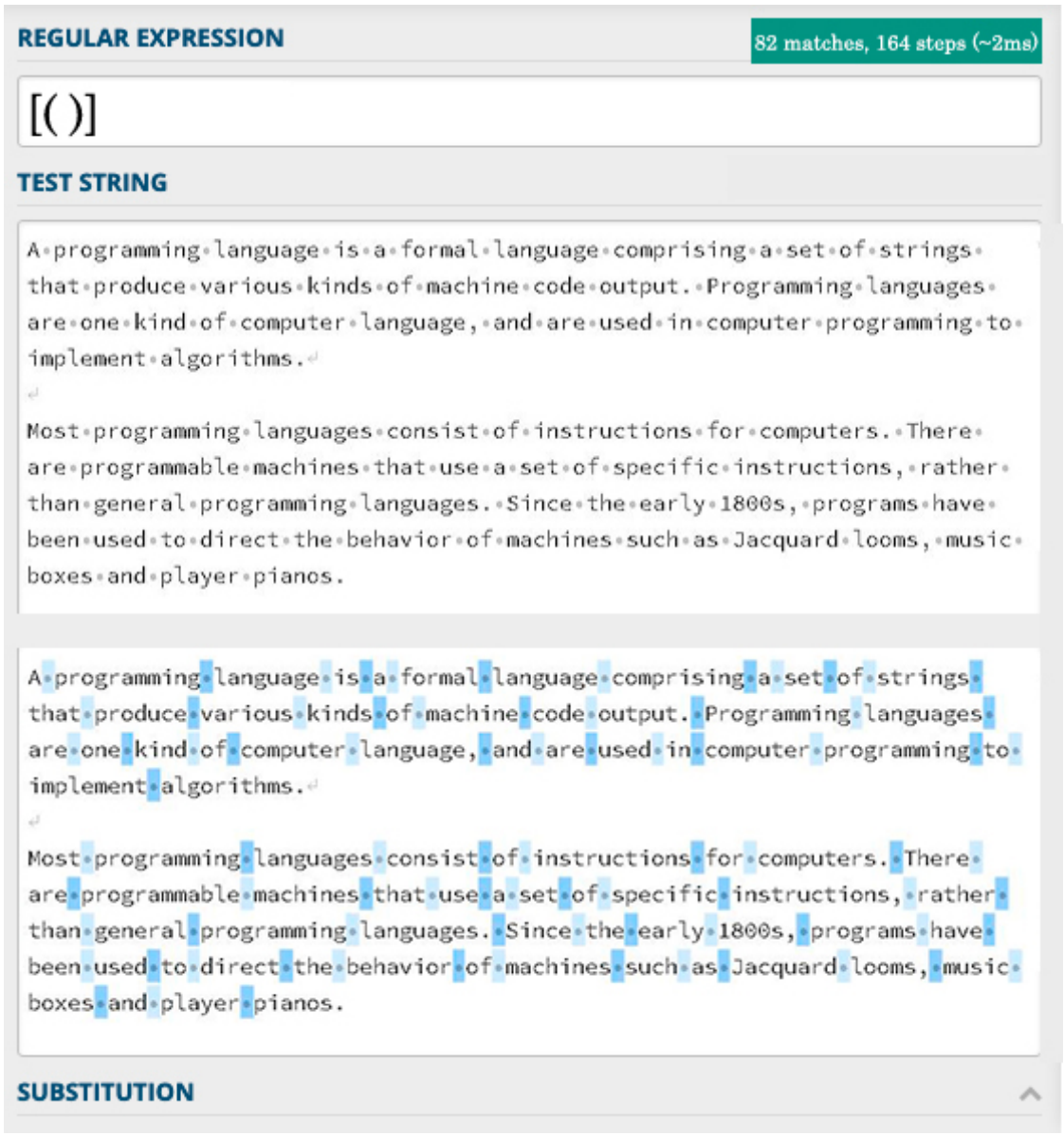
```
            return false;
        }
        return !is_null($result);
    }

    public function destroy($id) {
        try {
            $result = $this->db->execute("DELETE FROM
sessions WHERE id = ?", ["s", $id]);
        } catch (\Exception $ex) {
            return false;
        }
        return !is_null($result);
    }

    public function gc($max) {
        $old = time() - $max;

        try {
            $result = $this->db->execute("DELETE FROM
sessions WHERE access < ?", ["s", $old]);
        } catch (\Exception $ex) {
            return false;
```

```
        }

        return !is_null($result);

    }

}
```

Скріншоти роботи програми



Малюнок 6.1


REGULAR EXPRESSION
82 matches, 164 steps (~2ms)
[( )]
TEST STRING
A programming language is a formal language comprising a set of strings that produce various kinds of machine code output. Programming languages are one kind of computer language, and are used in computer programming to implement algorithms.
Most programming languages consist of instructions for computers. There are programmable machines that use a set of specific instructions, rather than general programming languages. Since the early 1800s, programs have been used to direct the behavior of machines such as Jacquard looms, music boxes and player pianos.
A programming language is a formal language comprising a set of strings that produce various kinds of machine code output. Programming languages are one kind of computer language, and are used in computer programming to implement algorithms.
Most programming languages consist of instructions for computers. There are programmable machines that use a set of specific instructions, rather than general programming languages. Since the early 1800s, programs have been used to direct the behavior of machines such as Jacquard looms, music boxes and player pianos.
SUBSTITUTION

Малюнок 6.2

7. Література

1. Джон Хопкрофт, Раджід Монтвані, Джеффрі Ульман. Введення в теорію автоматів, мов та обчислення. Москва, Санкт-Петербург, Київ 2002

2. Ю.Г. Карпов. Теорія автоматів. Санкт-Петербург 2003

3. D. Milnor On nonregular automata, Computotorica, 23 (2) (2011), 183-192.

4. Гілл Е. Линейные последовательные машины, Москва, 1988.