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 математичного забезпечення комп'ютерних систем

(повна назва кафедри (предметної, циклової комісії))

# Дипломна робота

*на здобуття освітньо-кваліфікаційного рівня бакалавр*

на тему Скінченні автомати

Finite automates

Виконав: студент денної форми навчання
напряму підготовки 6.050102 – Комп'ютерна інженерія .

(шифр і назва напряму підготовки, спеціальності)

Дубяго Ілля Васильович

(прізвище, ім'я, по-батькові)

Керівник доц. Сімонова І. Г.

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

Рецензент проф.Варбанець П.Д.

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

Рекомендовано до захисту:

Протокол засідання кафедри

Комп'ютерної алгебри та

Дискретної математики

№_____ від «___»_____ 2019 р.

Завідувач кафедри

_____ П.Д.Варбанець

(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК № ___

протокол № __ від «___» ____ 2019 р.

Оцінка _____/_____/_____

(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____ О.О.Арсірій

(підпис) (прізвище, ініціали)

Одеса - 2019

# АННОТАЦИЯ

В дипломной работе разрабатывается тема «Конечные автоматы».

Цель работы – исследование алгоритма построения конечных автоматов их декартового произведения, а также минимизация автомата полученного этим произведением.

В результате работы создано программное обеспечение, которое имитирует работу конечного автомата, строит объединение двух детерминированных конечных автоматов. Для минимизации, заданого автомата, осуществляется поиск недостижимых состояний, поиск эквивалентных состояний и строится минимальная форма конечного автомата.

# АНОТАЦІЯ

У дипломній роботі розробляється тема «Кінцеві автомати».

Мета роботи – дослідження алгоритму побудови кінцевих автоматів їх декартового добутку, а також мінімізація автомата отриманого цим добутком.

В результаті роботи створено програмне забезпечення, яке імітує роботу кінцевого автомата, будує об’єднання двох детермінованих кінцевих автоматів. Для мінімізації заданого автомата, здійснюється пошук недосяжних станів, пошук еквівалентних станів і будується мінімальна форма кінцевого автомата.

# ABSTRACT

In the thesis the theme "Finite automates" is being developed.

The aim of the paper is to study the algorithm of creation of finite-state machine of their cartesian work and also minimization of the automatic machine received by this work.

As a result of work the software which imitates operation of the finite-state machine is created, builds association of two deterministic finite-state machines. For minimization, the set automatic machine, search of unattainable statuses, search of equivalent statuses is carried out and the minimum form of the finite-state machine is under construction.

ЗМІСТ

# ВСТУП

Двадцяте століття стало часом прориву в багатьох аспектах науки і техніки. Одним з найбільш визначних досягнень того часу стало створення і широке використання електронних цифрових машин з програмним керуванням. Передумовами їх створення стали машина Поста і машина Тюрінга.

В 1935 році математики Пост і Тьюрінг дали перші уточнення щодо поняття "алгоритм".

Теорія алгоритмів будує і вивчає конкретні моделі алгоритмів. З розвитком обчислювальної техніки і теорії програмування зростає необхідність побудови нових економічних алгоритмів, змінюються способи їх побудови, способи запису алгоритмів на мові, зрозумілій виконавцю. Основний тип виконавця алгоритмів - комп'ютер, тому необхідно створювати спеціальні засоби що дозволяють з одного боку, розробнику у зручному вигляді записувати алгоритми, а з іншого - що дають комп'ютеру розуміти ці записи.

Візьмемо до прикладу А. Тюрінга і його машину. Його метою було точно описати кордон між тим, що обчислювальна машина може робити, і тим, чого вона не може. Отримані їм результати можна застосувати не тільки до абстрактних машинам Тюрінга, а й до реальних сучасним комп’ютерів.

У 1940-х і 1950-х роках багато дослідників займалися вивченням і побудовою найпростіших машин, які зараз називаються "скінченними автоматами". Вчені хотіли виставити такі автомати як моделі функціонування людського мозку. Але незабаром вони знайшли своє застосування і для безлічі інших цілей, які ми розглянемо пізніше. Необхідно також згадати дослідження лінгвіста Н.Хомського. В кінці 1950-х він займався вивченням формальних "граматик", які не будучи машинами в прямому сенсі слова, стали тісно пов'язаними з абстрактними автоматами та

виконують роль основи деяких найважливіших складових програмного забезпечення, зокрема, компонентів компіляторів.

У 1969 році с. Кук розвинув дослідження Тюрінга про межі обчислюваності та необчислюваності машин. Йому вдалося розділити завдання на ті, які можуть бути вирішені обчислювальною машиною швидко і ефективно, і ті, які в теорії, можуть бути вирішені, але для цього їм знадобиться так багато машинного часу, що комп'ютер стає марним для майже усіх экземплярів завдання, за винятком невеликих. Завдання такого класу називаються "трудновирішаємі" або "NP-важкими" [1]. Навіть при високому розвитку і зростанні швидкодії таких машин ймовірність що нам вдасться досягти успіхів у вирішенні завдань такого класу вельми мала.

Все це пов'язано з тим, чим займаються науковці в галузі інформатики та інформаційних технологій зараз. Багато з яких перерахованих вище понять, таких як формальні граматики чи скінченні автомати, застосовуються при проектуванні і створенні важливих та основних компонентів програмного забезпечення. А такі поняття, як наприклад, машина Тюрінга, пояснюють нам принципові можливості програмного забезпечення. Дослідження С.Кука про "трудновирішаємі" завдання допомагають визначити нам, чи можемо ми вирішувати те чи інше завдання безпосередньо і писати програму для вирішення її, чи нам необхідно продумати як вирішити таке завдання в обхід використовуючи метод за допомогою якого у нас вийде зменшити час, що витрачається програмою на її рішення.

Основною проблемою у спілкуванні людини і машини є те, що комп'ютер не розпізнає нашої мови (російську, англійську та інш.). Мову який розуміє комп'ютер це нулі та одиниці. Цей розрив у розумінні один одного долається за допомогою штучної мови, що дозволяє вживати певні слова, пропозиції та формули, які машина може "зрозуміти". Створюється програмне забезпечення, за допомогою якого обчислювальна машина приймає послідовність бітів, які представляють команди написані людиною на штучному мовою, і транслює їх у внутрішні бітові структури, необхідні

для виконання того, що хоче людина. Існує два види програм, що дозволяють машині розуміти пропозиції вхідної мови, використовуваної програмістом.

1. Інтерпретатор - програма допускає в якості входу вихідну програму, і виробляє обчислення, приписувані цією програмою.

2. Транслятор - програма, яка в якості входу бере програму написану на мові зрозумілій людині, а на вихід видає програму написану об'єктною мовою. Об'єктна мову зазвичай є машинною мовою обчислювальної машини, в цьому випадку програму можна відразу виконувати. Умовно транслятори поділяються на асемблери і компілятори, що транслюють мови низького і високого рівнів відповідно [1].

Основа побудови компіляторів лежить в теорії автоматів.

Актуальність теми. Як говорилося вище, скінченні автомати використовуються в багатьох аспектах програмного і апаратного забезпечення, ось найбільш важливі з них.

1. Програмне забезпечення для сканування та обробки великих текстів, таких як набори Web-сторінок, з метою пошуку заданих слів, речень та інш.

2. Програмне забезпечення, що застосовується при розробці та перевірці цифрових схем.

3. "Лексичний блок" стандартного компілятора. Це та частина компілятора яка відповідає за розмежування початкового тексту на такі частини як: ключові слова, ідентифікатори і знаки пунктуації.

4. Програмне забезпечення для перевірки таких систем як протоколи зв'язку або протоколи для захищеного обміну інформації. Ті системи, котрі можуть знаходитись у кінцевому числі різних станів [1].

Метою роботи є вивчення системи кінцевих автоматів і програмування цієї системи. Хоча теорія кінцевих автоматів вивчає дуже прості моделі, вона є фундаментом великої кількості різноманітних додатків. Ці додатки - від мовних процесів до систем управління реального часу і протоколів зв'язку - покривають значну частку систем, розробкою, реалізацією та аналізом яких займається інформатика.

Для досягнення поставленої цілі були сформульовані і виконані наступні задачі:

- проаналізовати основні проблеми предметної області;
- провести аналіз та обґрунтування методів реалізації;
- побудувати загальну архітектуру системи і обрати програмну платформу для її реалізації;
- виконати програмну реалізацію інформаційної системи
- перевірити працездатність програмної реалізації.

# 1 ПОСТАНОВКА ЗАДАЧІ

Для розкриття теми даної роботи необхідно:

1. Ознайомитись з теоріями кінцевих автоматів.
2. Виконання теоретико-множинних операцій над автоматними множинами.
3. Еквівалентність станів і автоматів.
4. Мінімізація кінцевих автоматів.
5. Програмне забезпечення завдань.

# 2 ТЕОРЕТИЧНІ МАТЕРІАЛИ

Перед тим як дати формальне поняття що таке скінченні автомати, коротко опишемо, що він собою являє. Кінцевий автомат складається з безлічі станів та "керування", переводить з одного стану в інший залежно від отриманих "вхідних даних". Число станів кінчено і запам'ятати всю історію системи неможливо, тому потрібно будувати систему ретельно, щоб зберігати тільки дійсно важливу інформацію, а непотрібну видалити. Перевага скінченності числа станів полягає в тому, що систему можна реалізувати, маючи обмежені ресурси. Наприклад, її можна реалізувати "в залізо" як схему або ж у вигляді простої програми, що приймає рішення на основі обмеженої кількості даних або поточної команди машинного коду. Також згадаємо про класи автоматів, які істотно відрізняються за типом цього управління. Управління може бути "детермінованим" у тому сенсі, що автомат може перебувати в кожний момент часу не більше ніж в одному стані, і "недетрмінованим", тобто автомат може одночасно знаходиться в декількох станах. Додавання недетермінизма не дозволяє визначати мови, які не можна було б визначити за допомогою детермінованих кінцевих автоматів. Але не дивлячись на це недетерміновані автомати виявляються доволі корисні в додатках. Саме недетермінизм дозволяє нам програмувати рішення завдання використовуючи мови високого рівня.

## 2.1 Визначення кінцевого автомата.

Кінцевим автоматом $\Omega$ називається п'ятірка $\langle A, B, Q, \delta, \lambda \rangle$, де $A, B$ - непорожні множини, називаються відповідно вхідним та вихідним алфавітом; $Q$ - непорожня кінцева множина, зване множиною внутрішніх станів автомата; $\delta : Q \times A \to Q$ - функція переходів; $\lambda : Q \times A \to B$ - функція виходу [4].

Якщо у множині $Q$ виділено певний стан $q_0$, званий початковим станом, то автомат $\Omega$ називається ініціальним автоматом і задається шісткою об'єктів, тобто $\Omega = (A, B, Q, q_0, \delta, \lambda)$.

Ініціальний КА завжди починає свою роботу з фіксованого стану $q_0$. Наприклад, якщо включити комп'ютер (ініціальний КА), то на екрані з'явиться певне зображення, точніше, однозначно певна послідовність картинок. При включенні телевізора (неініціального КА) з'являються різні картинки.

## 2.2 Неформальне описаня роботи кінцевого автомата

КА реалізує відображення множини слів вхідного алфавіту у множину слів вихідного алфавіту наступним чином. У початковий момент часу КА знаходиться в деякому стані $q_j$ (ініціальний КА починає роботу зі стану $q_0$). Вхідне слово $a_i = a_1\ a_2\ \dots\ a_n$ символами (літерами) подається на вхід КА. Залежно від вхідного символу $a_i$, і поточного стану $q_j \in Q$, КА видає вихідний символ і переходить у новий стан $q_{ij} \in Q$ (можливо, що співпадає зі станом $q_j$). Таким чином, послідовність символів вхідного слова однозначно визначає послідовність символів вихідного слова $b_i = b_1\ b_2\ \dots\ b_n$. При цьому говорять, що вихідне слово $b_i$ відповідає вхідному слову $a_i$.

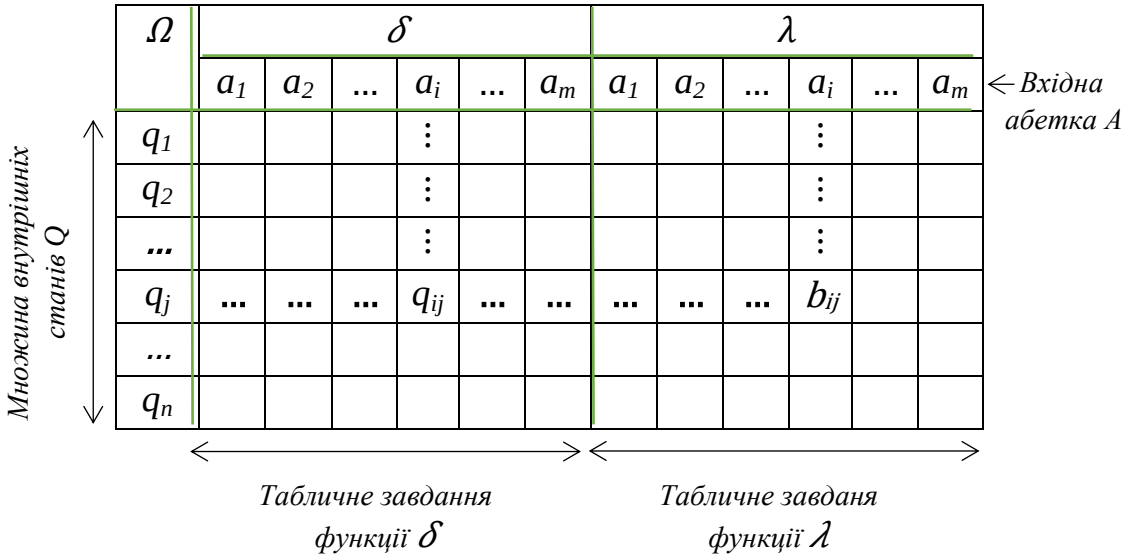
КА не має СТОП-стану, тобто стану, в якому він закінчує свою роботу: КА працює "до останнього клієнта", поки на вхід подаються символи (літери) вхідного слова.

## 2.3 Способи завдання кінцевого автомата

### 2.3.1 Табличний

Нехай $\Omega = (A, B, Q, \delta, \lambda)$, де $A, B, Q$ - множини; $\delta, \lambda$ - функції, задані на множині $Q \times A$. автомат задається таблицею, на вході якої перераховані

елементи множини $A$ (зверху, двічі) і $Q$ (зліва). Перша половина таблиці (таблиця переходів) задає функцію $\delta$, друга половина (таблиця виходів) - функцію $\lambda$ [5].




2.3.2 За допомогою діаграми Мура

Діаграма Мура для КА $\Omega=\langle A, B, Q, \delta, \lambda\rangle$ - це кінцевий розмічений орієнтований псевдограф, вершини якого взаємно однозначно відповідають станам автомата, а мітками ребер служать пари символів $a/b$, де перший символ належить до вхідного алфавіту, а другий - вихідному алфавіту. При цьому, якщо $a_i \in A$, $b_r \in B$, $q_j$, $q_k \in Q$ и $\delta(a_i,q_j)=q_k$, $\lambda(a_i,q_j)=b_r$, то з вершини $q_j$ у вершину $q_k$ йде орієнтоване ребро, позначене парою $a_i/b_r$.

Для того, щоб зазначений граф був графом переходів КА, необхідно і достатньо, щоб виконувалися дві умови:

1. $\forall q_j \in Q$ та $\forall q_i \in A$ існує ребро, що виходить з вершини $q_j$, мітка якого містить символ $q_i$ (умова повноти).

2. $\forall q_j \in Q$ и $\forall q_i \in A$ існує тільки одне ребро, що виходить з вершини $q_j$, мітка якого містить символ $q_i$ (умова однозначності) [2].

2.4 Розширення функцій $\delta, \lambda$. Автоматний оператор.

Для КА $\Omega = (A, B, Q, \delta, \lambda)$ можна визначити функції переходів і виходу не тільки на множині $Q \times A$, але і на множині $Q \times A^*$. Позначимо ці функції $\Delta$ і $\Lambda$ відповідно.

Рекурсивне визначення функції $\Delta$:

$$(\forall q \in Q)(\forall a \in A)(\forall \omega \in A^*)\begin{cases} \Delta(q, \wedge) = q; \\ \Delta(q, \omega a) = \delta(\Delta(q, \omega), a). \end{cases}$$

Змістовний сенс функції $\Delta$: $\forall q \in Q$ та $\forall \omega \in A^*$ значенням $\Delta(q,\omega)$ є стан, у який перейде КА після обробки вхідного слова $\omega$ за умови, що автомат починає свою роботу зі стану $q$.

Рекурсивне визначення функції $\Lambda$:

$$(\forall q \in Q)(\forall a \in A)(\forall \omega \in A^*)\begin{cases} \Lambda(q, \wedge) = \wedge; \\ \Lambda(q, \omega a) = \lambda(\Delta(q, \omega), a). \end{cases}$$

Змістовний сенс функції $\Lambda$: $\forall q \in Q$ та $\forall \omega \in A^*$ значенням $\Lambda(q,\omega)$ є останній символ вихідного слова, відповідний вхідному слову $\omega$ за умови, що автомат починає свою роботу зі стану $q$.

Рекурсивне визначення функції $\Omega : Q \times A^* \to B^*$:

$$(\forall q \in Q)(\forall a \in A)(\forall \omega \in A^*)\begin{cases} \Omega(q, \wedge) = \wedge; \\ \Omega(q, \omega a) = \Omega(q, \omega)\lambda(\Delta(q, \omega), a). \end{cases}$$

Функція $\Omega$ називається автоматним оператором кінцевого автомата $\Omega$.

Змістовний сенс функції $\Omega$ $\forall q \in Q$ та $\forall \omega \in A^*$ значенням $\Omega(q,\omega)$ є вихідне слово, відповідне профілю вхідного слова $\omega$ за умови, що автомат починає свою роботу зі стану $q$.

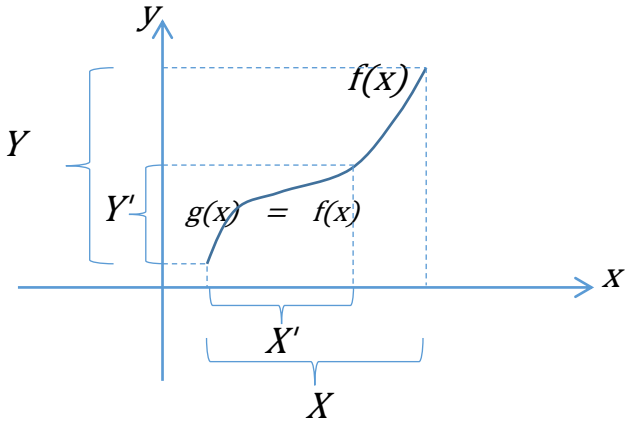
Визначення функцій $\Delta$, $\Lambda$, $\Omega$ можна наочно інтерпретувати на графі переходів КА $\Omega$. Якщо автомат починає свою роботу зі стану $q$, то всяке вхідне слово $a_1\ a_2\ \ldots\ a_k$ однозначно визначає маршрут довжини $k$, що починається у вершині $q$. Значення функції $\Delta(q,\ a_1\ a_2\ \ldots\ a_k)$ равно останній

вершині цього маршруту. Значення $\Lambda(b_1\ b_2\ \dots\ b_k)$ - це символ вихідного алфавіту $b_k$, який відзначає останнє ребро цього маршруту. Значення функції $\Omega\ (q,\ a_1\ a_2\ \dots\ a_k)$ - це вихідне слов $b_1\ b_2\ \dots\ b_k$, складене з нижніх символів міток ребер зазначеного маршруту.

Щоб уточнити зв'язок між функціями $\delta$, $\lambda$ і $\Delta$, $\Lambda$ відповідно, згадаємо визначення розширення функції.

Нехай $f: X \to Y$ та $g: X' \to Y$ - функції, задані на множинах $X$ та $X'$ відповідно. Функція $f$ називається розширенням функції $g$, якщо виконуються такі умови:

1. $X \subseteq X'$;
2. $Y \subseteq Y'$;
3. $\forall x \in X': f(x) = g(x)$.




Теорема 2.4.1 (про розширеня функций $\delta$ и $\lambda$).

Нехай задано КА $\Omega = (A, B, Q, \delta, \lambda)$. Тоді:

1. Функція $\Delta$ є розширенням функції $\delta$;
2. Функція $\Lambda$ є розширенням функції $\lambda$;

Доведення.

1. $\left.\begin{matrix}\delta: Q \times A \to Q \\ \Delta: Q \times A^* \to Q\end{matrix}\right\} \Rightarrow \begin{cases} Q \times A \subseteq Q \times A^* \\ Q \subseteq Q \end{cases}$, тобто виконуються перші дві умови з ухвали розширення функції. Доведемо, що виконується третя умова

зазначеного визначення, тобто доведемо, що на словах одиничною довжини значення функцій $\delta$ і $\Delta$ збігаються. Дійсно, враховуючи, що $\Delta(q,\wedge)=q$, маємо:

$$(\forall q\in Q)(\forall a\in A)\ \Delta(q,a)=\delta(\Delta(q,\wedge),a)=\delta(q,a).$$

2. $\left.\begin{matrix}\lambda: Q\times A\to B\\ \Lambda: Q\times A^*\to B\end{matrix}\right\}\Rightarrow\begin{cases}Q\times A\subseteq Q\times A^*\\ B\subseteq B\end{cases}$, тобто виконуються перші дві умови з ухвали розширення функції. Доведемо, що виконується третя умова зазначеного визначення, тобто доведемо, що на словах одиничною довжини значення функцій $\lambda$ і $\Lambda$ збігаються [2]. Дійсно, враховуючи, що $\Delta(q,\wedge)=q$, маємо:

$$(\forall q\in Q)(\forall a\in A)\Lambda(q,a)=\lambda(\Delta(q,\wedge),a)=\lambda(q,a).$$

Лемма 2.4.2 (властивості функцій $\Delta$ и $\Omega$).

Нехай задано КА $\Omega=(A, B, Q, \delta, \lambda)$. Тоді $\forall\omega,\vartheta\in A^*$ та $\forall q\in Q$ мають місце рівності:

1. $\Delta(q, \omega\vartheta)=\Delta(\Delta(q,\omega),\vartheta)$;
2. $\Omega(q, \omega\vartheta)=\Omega(q, \omega)\,\Omega((\Delta(q, \omega), \vartheta)$.

Доведення. Проведемо доказ першого рівності індукцією по довжині слова $\vartheta$.

1. База індукції. Нехай $|\vartheta|=0$. Тоді $\vartheta=\wedge$ та $\Delta(\Delta(q, \omega),\wedge)=\Delta(q, \omega)$.
2. Припущення індукції. Припустимо, що рівність $\Delta(q,\omega\vartheta)=\Delta(\Delta(q,\omega),\vartheta)$ виконується для будь-якого вхідного слова $\vartheta$, довжина котрого $|\vartheta|\le k$.
3. Крок індукції. Нехай $|\vartheta|=k+1$. Тоді $\vartheta=\vartheta' a$, де $|\vartheta'|=k$ та $a\in A$.

Маємо:

$$\Delta(q, \omega\vartheta)=\Delta(q, \omega\vartheta' a)\overset{def\ \Delta}{=}\delta(\Delta(q, \omega\vartheta'),a)\overset{Предположение\ индукции}{=}\delta(\Delta(\Delta(q, \omega),\vartheta'),a)$$
$$\overset{def\ \Delta}{=}\Delta(\Delta(q, \omega),\vartheta' a)=\Delta(\Delta(q, \omega),\vartheta).$$

Крок індукції доведений і, тим самим, перше рівенство $\Delta(q, \omega\vartheta)=\Delta(\Delta(q,\omega),\vartheta)$ доведено.

Проведемо доказ другої рівності індукцією по довжині слова $\vartheta$.

1. База індукції. Нехай $|\vartheta|=0$. Тоді $\vartheta=\wedge$ та

$\Omega(q, \omega\wedge)=\Omega(q, \omega)\,\Omega((\Delta(q, \omega), \wedge)=\Omega(q, \omega)\wedge=\Omega(q, \omega).$

2. Припущення індукції. Припустимо, що рівність $\Omega(q, \omega\vartheta)=$ $=\Omega(q, \omega)\,\Omega((\Delta(q, \omega), \vartheta)$. виконується для будь-якого вхідного слова $\vartheta$, довжина котрого $|\vartheta|\le k$.

3. Крок індукції. Нехай $|\vartheta|=k+1$. Тоді $\vartheta=\vartheta' a$, де $|\vartheta'|=k$ та $a\in A$.

Маємо:

$$\Omega(q, \omega\vartheta)=\Omega(q, \omega\vartheta' a) \overset{def\ \Omega}{=} \Omega(q, \omega\vartheta')\,\lambda(\Delta(q, \omega\vartheta'), a) \overset{Предположение\ индукции}{=}$$

$$\Omega(q, \omega)\Omega(\Delta(q, \omega),\vartheta')\,\lambda(\Delta(q, \omega\vartheta'), a) \overset{равенство\ 1.}{=}$$

$$\Omega(q, \omega)\,\Omega(\underbrace{\Delta(q, \omega)}_{q'},\vartheta')\,\lambda(\Delta(\underbrace{\Delta(q, \omega)}_{q'},\vartheta'), a)=$$

$$=\Omega(q, \omega)\,\Omega(q',\vartheta')\,\lambda(\Delta(q',\vartheta'), a) \overset{def\ \Omega}{=}$$

$$\overset{def\ \Omega}{=} \Omega(q, \omega)\Omega(q',\vartheta' a)=\Omega(q, \omega)\Omega(\Delta(q,\omega),\vartheta).$$

Крок индукції доведено, таким чином, друге рівності $\Omega(q, \omega\vartheta)=$ $=\Omega(q, \omega)\,\Omega((\Delta(q, \omega), \vartheta)$, доведено.

Теорема 2.4.3 (о детермінованності КА).

Нехай задан КА $\Omega=(A, B, Q, \delta, \lambda)$. Зафіксуємо певний стан $q\in Q$. Тоді $S=(A^*, B^*, \Omega(q, \vartheta))$ – детермінована система [2].

Доведення. Система S - словникова функціональна система.

Проведемо доказ індукцією по довжині вхідного слова, що $\forall\vartheta\in A^*:|\vartheta|=|\Omega(q, \vartheta)|$.

1. База індукції. Нехай $|\vartheta|=0$. Тоді $\vartheta=\wedge$ та $|\Omega(q, \wedge)|=|\wedge|=0$.

2. Припущення індукції. Припустимо, що рівність $|\vartheta|=|\Omega(q, \vartheta)|$ виконується для будь-якого вхідного слова $\vartheta$, довжина котрого $|\vartheta|\le k$.

3. Крок індукції. Нехай $|\vartheta|=k+1$. Тоді $\vartheta=\vartheta' a$, де $|\vartheta'|=k$ та $a\in A$.

Маємо:

$|\Omega(q, \vartheta)| = |\Omega(q, \vartheta' a)| = |\Omega(q, \vartheta')\lambda(\Delta(q, \vartheta'), a)| = |\Omega(q, \vartheta')| + 1 = |\vartheta'| + 1 = |\vartheta|$.

Крок доведений і, тим самим, перша умова детермінованої системи обгрунтована.

Доведемо, що $\forall u, w \in A^*: [u \preccurlyeq w \Rightarrow \Omega(q, u) \preccurlyeq \Omega(q, w)]$. $u \preccurlyeq w \Rightarrow \exists v \in A^* [w = uv]$.

Тоді: $\Omega(q, w) = \Omega(q, uv) = \Omega(q, u)\,\Omega(\Delta(q, u), v)$. Таким чином $\Omega(q, u) \preccurlyeq \Omega(q, w)$ та друга умова детермінованої системи обгрунтована. Теорема доведена повністю.

Теорема 2.4.4 (про існування КА для ДС).

Для всякої детермінованою системи (ДС) $S = (A^*, B^*, F)$ з кінцевою множиною $A^*/\sim_F$ (де $\sim_F$ розуміється в сенсі еквівалентності Майхил-Нероуда) можна побудувати кінцевий автомат

$\Omega_F = (A, B, Q_F, q_{oF}, \delta_F, \lambda_F)$ з тим же виходом, тобто $\forall w \in A^*: F(w) = \Omega_F(q_o, w)$.

Доведення.

Вхідной та вихідной алфавіти КА $\Omega_F$ збігаються відповідно з вхідним та вихідним алфавітами ДС. В якості множини внутрішніх станів КА $\Omega_F$ будемо розглядати фактор множини множини $A^*$ по відношенню «$\sim_F$», тобто $Q_F = A^*/\sim_F$.

Клас еквівалентності, відповідний порожньому слову, буде грати роль початкового стану ($q_{oF} = [\wedge]_{\sim_F}$).

Функцію переходів $\delta_F : Q_F \times A \to Q_F$ (тобто $\delta_F : (A^*/\sim_F) \times A \to (A^*/\sim_F)$) визначимо наступним чином:

$$(\forall v \in A^*)(\forall a \in A)\{\delta_F([v] \sim_F, a) = [va] \sim_F\}.$$

Доведемо, що функція δF визначена коректно. Маємо

$$(\forall v,v'\in A^*)(\forall a\in A): (v \sim_F v' \overset{def\sim_F}{\Leftrightarrow} f_v=f_{v'}) \overset{\text{свойство 3 остаточных функций}}{\Rightarrow} f_{va}=f_{v'a} \overset{def\sim_F}{\Leftrightarrow}$$

$$va \sim_F v'a \overset{def [v]_{\sim_F}}{\Rightarrow} [va]_{\sim F}=[v'a]_{\sim F}.$$

Функція виходів $\lambda_F$: $Q_F \times A \to B$ (тобто $\lambda_F$: $(A^*/\sim_F)\times A \to B$) визначимо наступним чином:

$$(\forall v\in A^*)(\forall a\in A)\{\lambda_F[v]\{\lambda_F[v]\sim_F,a)= f_v(a)\}$$

Для КА $\Omega_F$ функції $\Delta_F$, $\Lambda_F$, $\Omega_F$ визначаються стандартним чином.

Доведемо, що функція $\Delta_F$ має можливості

$$(*) \qquad \forall v\in A^*: \Delta_F(q_o,v)= [v]\sim_F.$$

Доведемо індукцією по довжині вхідного слова $v$.

1. База індукції. Нехай $|v|=0$, Тоді $v=\wedge$ та

$$\Delta_F(q_o,\wedge) = \Delta_F([\wedge]\sim_F, \wedge) =[\wedge\wedge]\sim_F = [\wedge]\sim_F$$

2. Припущення індукції. Припустимо, що рівність $\Delta_F(q_o,v)= [v]\sim_F$ виконується для будь-якого вхідного слова $v$, Довжина якого $|v|\le k$.

3. Крок індукції. Нехай $|v|=k+1$. Тоді $v = v'a$, де $|v'|=k$ та $a\in A$. Маємо:

$$\Delta_F(q_o,v)=\Delta_F(q_o,v'a) \overset{def\Delta_F}{=} \delta_F(\Delta(q_o,v'),a) \overset{\text{предположение индукции}}{=} \delta_F([v']\sim_F,a) \overset{def\delta_F}{=} [v'a]\sim_F=[v']\sim_F.$$

Крок доведений отже, властивість (*) обґрунтовано.

Використовуючи властивість (*) функції $\Delta_F$, доведемо, що $\forall v\in A^*$: $F(v)=$

$= \Omega_F(q_o,v)$.

Доказ проведемо індукцією по довжині вхідного слова $v$.

1. База індукції. Нехай $|v|=0$. Тоді $v=\wedge$ та

$$\left.\begin{array}{l} F(\wedge)=\wedge; \\ \Omega_F(q_0,\wedge)=\wedge \end{array}\right\} \Rightarrow \quad F(\wedge)=\Omega_F(q_0,\wedge)$$

.

2. Припущення індукції. Припустимо, що рівність $F(v)=\Omega_F(q_o,v)$ виконується для будь-якого вхідного слова $v$, довжина якого $|v|\le k$.

3. Крок індукції. Нехай $|v|=k+1$. Тоді $v = v'a$, де $|v'|=k$ и $a\in A$.

Маємо:

$$\Omega_F(q_0,v) = \Omega_F(q_0,v'a) \overset{def\ \Omega_F}{=} \Omega_F(q_0,v')\lambda_F(\Delta_F(q_0,v'),a) \overset{\text{предположение индукции}}{=} F(v')\lambda_F(\Delta_F(q_0,v'),a)$$

$$\overset{\text{свойство}(*)}{=} F(v')\lambda_F([v'],a) \overset{def\ \lambda_F}{=} F(v')f_{v'}(a) \overset{def\ f_v}{=} F(v'a) = F(v).$$

Крок індукції доведений отже, $\forall v \in A^*: F(v) = \Omega_F(q_0,v)$.

## 2.5 Класи кінцевих автоматів.

Автомат $\Omega = \langle A, B, Q, \delta, \lambda \rangle$ називається автоматом з короткою пам'ять, якщо $(\forall u,v \in A^*)(\exists k \in \mathbb{N})(\forall w \in A^*)[|w|>k \Rightarrow \Lambda(q_0,uw) = \Lambda(q_0,vu)$.

Прикладом автомата з короткою пам'ятю є КА, що реалізує додавання двійкових чисел.

Автомат $\Omega = \langle A, B, Q, \delta, \lambda \rangle$ називається генератором, якщо $|A| = 1$, тобто якщо вхідний алфавіт складається тільки з одного символу.

Автомат $\Omega = \langle A, B, Q, \delta, \lambda \rangle$ називається акцептором (або автоматом-розпізнавачем), якщо його вихідний алфавіт $B = \{0, 1\}$.

Автомат-затримка - це автомат, який, отримавши наступний символ, видає попередній [4].

Автомат-компаратор - це автомат, на вхід якого одночасно по символу подаються два вхідних слова. Поки що слова збігаються, автомат видає одиниці, а після першого неспівпадіння автомат видає тільки нулі.

Автомат-акцептор або автомат-распознаватель може бути представлений у вигляді автомата-сейфа або у вигляді автомата-жаби.

## 2.6 Еквівалентність станів та автоматів.

Нехай задані: $\Omega_1 = \langle A, B, Q_1, \delta_1, \lambda_1 \rangle$ та $\Omega_2 = \langle A, B, Q_2, \delta_2, \lambda_2 \rangle$ з вхідними і вихідними алфавітами що збігаються [5].

Стан $q_i$ автомата $\Omega_1$ називається $k$-еквівалентним станом $q_j$ автомата

$\Omega_2$ (позначення: $q_i \sim_k q_j$), якщо $(\forall w \in A^* \wedge |w|=k)[\Omega_1(q_i,w) = \Omega_2(q_j,w)]$.

Якщо стани $q_i$ та $q_j$, не є $k$-еквівалентними, то вони називаються $k$-різними (позначення: $q_i \nsim_k q_j$).

Стани $q_i$ та $q_j$, $k$-еквівалентні при будь-якому $k$, називаються еквівалентними (позначення $q_i \sim q_j$).

Надалі, говорячи про $k$-еквівалентності або про еквівалентність станів КА, ми не завжди будемо підкреслювати, що мова може також і про кількох автоматів одночасно. Цей факт слід враховувати при читанні подальшого матеріалу.

Теорема 2.6.1 (про відношення k-эквівалентності (эквівалентності)).

Відношення $k$-еквівалентності (еквівалентності) на множині станів кінцевого автомата (на декартовому добутку множини станів кінцевої сукупності КА із загальними вхідними і вихідними алфавітами) є відношенням еквівалентності, тобто рефлексивно, симетрично і транзитивно.

Слідство 2.6.1 Відношення $k$-еквівалентності визначає розбиття множини станів КА на попарно непересічні класи k-еквівалентних між собою станів [2].

Розбиття множини станів КА на класи $k$-евівалентних станів будемо позначати $\pi$ і називати $\pi_k$-розбиттям. Елементи цього розбиття (класи $k$-еквівалентних статків) будемо позначати $K_i$. Таким чином, $\pi_k=\{K_1, K_2, \dots, K_s\}$. Розбиття безлічі станів КА на класи еквівалентних станів будемо позначати $\pi$ і називати $\pi$-розбиттям.

Теорема 2.6.2 (властивості відносини $\sim_k$).

Нехай задані: $\Omega_1 = \langle A, B, Q_1, \delta_1, \lambda_1 \rangle$ та $\Omega_2 = \langle A, B, Q_2, \delta_2, \lambda_2 \rangle$ (можливо що збігаються). Тоді $\forall q_1 \in Q_1$ и $\forall q_2 \in Q_2$ справедливі наступні твердження:

1. $q_1 \sim_k q_2 \wedge 1 \le m \le k \Rightarrow q_1 \sim_m q_2$;

2. $q_1 \not\sim_k q_2 \wedge m \ge k \Rightarrow q_1 \not\sim_m q_2$;

3. $q_1 \sim_1 q_2 \Leftrightarrow$ рядки виходів для $q_1$ і $q_2$ в таблиці станів збігаються.

Доведення.

1. Нехай $w \in A^*$ и $|w| = m$. Виберемо довільне слово $v \in A^*$ таке, що $|v| = k - m$. Очевидно, що $|wv| = m + (k - m) = k$. Маємо:

$$q_1 \sim k\, q_2 \;\Rightarrow\; \Omega_1(q_1, wv) = \Omega_2(q_2, wv) \overset{\text{по свойству функции } \Omega}{\Rightarrow}$$

$$\Omega_1(q_1, w)\,\Omega_1(\Delta_1(q_1, w), v) = \Omega_2(q_2, w)\,\Omega_2(\Delta_2(q_2, w), v).$$

$$\left.\begin{array}{l} |\Omega_1(q_1, w)| = |w| = m \\ |\Omega_2(q_2, w)| = |w| = m \end{array}\right\} \Rightarrow \Omega_1(q_1, w) = \Omega_2(q_2, w) \overset{def\ \sim_k}{\Rightarrow} q_1 \sim k\, q_2.$$

2. Нехай $q_1 \not\sim_k q_2$. Тоді $(\exists w \in A^* \wedge |w| = k)[\Omega_1(q_1, w) \ne \Omega_2(q_2, w)]$. Якщо $v \in A^*$ та $|v| = m - k$, то $|wv| = k + (m - k) = m$. Маємо:

$$\Omega_1(q_1, wv) = \Omega_1(q_1, w)\,\Omega_1(\Delta_1(q_1, w), v);$$

$$\Omega_2(q_2, wv) = \Omega_2(q_2, w)\,\Omega_2(\Delta_2(q_2, w), v);$$

$\Omega_1(q_1, w) \ne \Omega_2(q_2, w)$, при цьому $|\Omega_1(q_1, w)| = |m| = |\Omega_2(q_2, w)|$.

Отже, $\Omega_1(q_1, w)\,\Omega_1(\Delta_1(q_1, w), v) \ne \Omega_2(q_2, w)\,\Omega_2(\Delta_2(q_2, w), v)$.

Таким чином, $q_1 \not\sim_m q_2$.

3. Справедливість цього твердження випливає з ухвали 1-еквівалентності на множині $A^*$.

Теорема 2.6.3 (про $(k+1)$-помітних станах).

Нехай задані: $\Omega_1 = \langle A, B, Q_1, \delta_1, \lambda_1 \rangle$ та $\Omega_2 = \langle A, B, Q_2, \delta_2, \lambda_2 \rangle$ (можливо що збігаються) $q_1 \in Q_1$ та $q_2 \in Q_2$.

Тоді: $(q_1 \sim_k q_2) \wedge (q_1 \not\sim_{k+1} q_2) \Rightarrow (\exists a \in A)[\delta_1(q_1, a) \not\sim_k \delta_2(q_2, a)]$

Іншими словами, стан, що належить одному класу $\pi$ розбиття, потрапляють у різні класи $\pi_{k+1}$-розбиття тільки тоді, коли знайдеться вхідна літера, котра переводить їх під дією функцій переходів в стану різних класів

$\pi k$-розбиття [2].

Доведення.

Доведемо контрапозицію:

$$(q_1 \sim_k q_2) \wedge (\forall a \in A)[\delta_1(q_1,a) \sim_k \delta_2(q_2,a)] \Rightarrow (q_1 \sim_{k+1} q_2).$$

Розглянемо довільне слово $w \in A^*$, $|w|=k$. Тоді $|aw|=k+1$. Маємо:

$$\Omega_1(q_1, aw) = \Omega_1(q_1, a)\, \Omega_1(\Delta_1(q_1, a), w) = \Omega_1(q_1, a)\, \Omega_1(\delta_1(q_1, a), w);$$

$$\Omega_2(q_2, aw) = \Omega_2(q_2, a)\, \Omega_2(\Delta_2(q_2, a), w) = \Omega_2(q_2, a)\, \Omega_2(\delta_2(q_2, a), w);$$

За умовою $(q_1 \sim_k q_2)$ и $\delta_1(q_1,a) \sim_k \delta_2(q_2,a)$. Тоді: $\Omega_1(q_1, a) = \Omega_2(q_2, a)$ та $\Omega_1(\delta_1(q_1, a), w) = \Omega_2(\delta_2(q_2, a), w)$. Відтак:

$$\Omega_1(q_1, a)\, \Omega_1(\delta_1(q_1, a), w) = \Omega_2(q_2, a)\, \Omega_2(\delta_2(q_2, a), w),$$

тобто: $\Omega_1(q_1, aw) = \Omega_2(q_2, aw)$. Звідки витікає, що: $(q_1 \sim_{k+1} q_2)$.

Таким чином контрапозиція а разом з нею теорія 2.6.3 доведена.

Слідство 2.6.3. Для будь-якого КА $\Omega = \langle A, B, Q, \delta, \lambda \rangle$ кількість класів $\pi_{k+1}$-розбиття не менша від кількості класів πk-розбиття, тобто $|\pi_{k+1}| \geq |\pi_k|$.

Доведення. При переході від $\pi_k$-розбиття до $\pi_{k+1}$-розбиття чинності теореми 2.6.3. кожен клас $\pi_k$-розбиття або цілком переходить у $\pi_{k+1}$-розбиття (якщо відповідна вхідна літера, котра переводить деякі стани цього класу під дією функцій переходів в стани різних класів $\pi_k$-розбиття, не знайдеться), або розпадється на нові класи $\pi_{k+1}$-розбиття (що збільшує кількість класів $\pi_{k+1}$-розбиття). Тому $|\pi_{k+1}| \geq |\pi_k|$ все доведено [2].

Слідство 2.6.4. Для будь-якого КА $\Omega = \langle A, B, Q, \delta, \lambda \rangle$: якщо $\pi_{k+1} = \pi_k$, то $\pi_k = \pi$ (тобто класи $k$-еквівалентних станів збігаються з класами еквівалентних станів).

Доведення. Якщо $\pi_{k+1} = \pi_k$, то при переході від $\pi_k$ до $\pi_{k+1}$ не відбулося дроблення класів отже, в силу теореми 2.6.3., не відбудеться їхнього дроблення при переході до наступним πm-розбиттям. Таким чином, отримаємо: $\pi_k = \pi_{k+1} = \pi_{k+2} = \ldots = \pi_m = \ldots$. Таким чином, будь-які $k$-

еквівалентні стани залишаються m-еквівалентними для будь-яких $m \geq k$. Оскільки ще $(\forall m<k)[q_i \sim_k q_j \Rightarrow q_i \sim_m q_j]$, то, в підсумку, будь-які $k$-еквівалентні стани залишаються m-еквівалентними для будь-яких $m$, тобто еквівалентними. Звідки витікає, що $\pi_k = \pi$ і все доведено.

Можна проведений вище доказ провести в більш розгорнутій формі. Доведемо її. Рівність $\pi_k = \pi_{k+1}$ означає, що для будь-яких двох станів $q_i$ і $q_j$, що належать одному класу еквівалентності розбиття $\pi_k$, і $\forall a \in A$ стану $\delta(q_i,a)$ та $\delta(q_j,a)$ також потрапляють до одного класу еквівалентності розбиття $\pi_k$ (в загальному випадку не що співпадає з вихідним), тобто, $\delta(q_i,a) \sim_k \delta(q_j,a)$. Формально ці міркування можна записати наступним чином. Нехай $\pi_k = \pi_{k+1}=\{K_1,K_2, \ldots ,K_s\}$, де $K_i$ - клас k-еквівалентних станів.Тоді

$$(\forall q_i, q_j \in Q)\{\{[(\exists K_l \in \pi_k)\{q_i, q_j \in K_l\} \Rightarrow (\forall a \in A)(\exists K_t \in \pi_k)\{\delta(q_i,a) \sim_k \delta(q_j,a)\}]\} \Rightarrow$$
$$\Rightarrow (q_i \sim_k q_j)\}.$$

Доведемо, що: $(\forall w \in A^*)\ (\forall q_i,q_j \in Q)[(q_i \sim_k q_j) \Rightarrow (\Delta(q_i,w) \sim_k \Delta(q_j,w))]$.

Доведення проведемо індукцією по довжині вхідного слова $w$.

База індукції. Нехай $|w|=0$. Тоді $w = \wedge$. Маємо

$$\begin{cases} \Delta(q_i, w) = \Delta(q_i, \wedge) = q_i \\ \Delta(q_j, w) = \Delta(q_j, \wedge) = q_j \end{cases} \Rightarrow [(q_i \sim_k q_j) \Rightarrow (\Delta(q_i, \wedge) \sim_k \Delta(q_j, \wedge)].$$

Припущення індукції. Нехай:

$$(\underset{|w| \leq n}{\forall} w \in A^*)[\Delta(q_i, w) \sim_k \Delta(q_j, w)].$$

Крок індукції. Нехай $|w| = n+1$. Тоді слово w можна представити у вигляді $w=w'a$, где $w' \in A^*$, $|w'| = n, a \in A$. Маємо:

$$\Delta(q_i, w) = \Delta(q_i, w'a) = \delta(\Delta(q_i, w'), a)$$

Чинності припущення індукції $\Delta(q_i,w') = \Delta(q_i,w')$. і тоді, в силу $\pi_k = \pi_{k+1}$, маємо: $\delta(\Delta(q_i, w'), a) \sim_k \delta(\Delta(q_j, w'), a)$. Отже $\Delta(q_i,w) = \Delta(q_i,w)$. і крок індукції доведений, значить, доведено твердження.

Доведемо, що якщо $\pi_k = \pi_{k+1}$, тоді $\pi_k = \pi$. По суті потрібно довести, що

$$(\forall q_i, q_j \in Q)[\pi_k = \pi_{k+1} \wedge q_i \sim_k q_j \Rightarrow q_i \sim q_j]$$

або, що те ж саме,

$$(\forall q_i, q_j \in Q)[\pi_k = \pi_{k+1} \wedge q_i \sim_k q_j \Rightarrow (\forall w \in A^*)\{\Omega(q_i, w) = \Omega(q_j, w)\}].$$

Доведення проведемо знову індукцією по довжині вхідного слова *w*. Нехай $\pi_k = \pi_{k+1}$ та. $q_i \sim_k q_j$

База індукції. Нехай $|w|=0$. Тоді $w = \wedge$. Маємо:

$$\begin{cases} \Omega(q_i, w) = \Omega(q_i, \wedge) = \wedge \\ \Omega(q_j, w) = \Omega(q_j, \wedge) = \wedge \end{cases} \Rightarrow [(q_i \sim_k q_j) \Rightarrow (\Omega(q_i, \wedge) \sim_k \Omega(q_j, \wedge)].$$

Припущення індукції. Нехай $(\underset{|w|=n}{\forall} w \in A^*)[\Omega(q_i, w) = \Omega(q_j, w)]$.

Крок індукції. Покладемо $|w| = n+1$. Тоді слово w можна представити у вигляді $w=w'a$, где $w' \in A^*$, $|w'| = n, a \in A$. Маємо:

$$(*) \quad \begin{cases} \Omega(q_i, w) = \Omega(q_i, w'a) = \Omega(q_i, w')\lambda((\Delta(q_i, w'), a); \\ \Omega(q_j, w) = \Omega(q_j, w'a) = \Omega(q_j, w')\lambda((\Delta(q_j, w'), a). \end{cases}$$

За припущенням індукції, $\Omega(q_i, w') = \Omega(q_j, w')$. По доведеному раніше $\Delta(q_i, w) \sim_k \Delta(q_i, w)$. Звідси одразу випливає, що $\Delta(q_i, w) \sim_l \Delta(q_i, w)$. Тоді з (∗): $\Omega(q_i, w') = \Omega(q_j, w')$ і, тим самим, крок індукції доведений.

Висновок індукції. $(\forall w \in A^*)[\Omega(q_i, w) = \Omega(q_j, w)]$ тобто, $q_i \sim q_j$.

Отже $(\forall q_i, q_j \in Q)[\pi_k = \pi_{k+1} \wedge q_i \sim_k q_j \Rightarrow q_i \sim q_j]$.

Скінченні автомати: $\Omega_1 = \langle A, B, Q_1, \delta_1, \lambda_1 \rangle$ та $\Omega_2 = \langle A, B, Q_2, \delta_2, \lambda_2 \rangle$ що збігаються вхідними і вихідними алфавітами називаються еквівалентними (позначення $\Omega_1 \sim \Omega_2$), якщо для кожного стану першого автомата знайдеться еквівалентний йому стан у другому автоматі і навпаки, тобто:

$$(\forall q_1 \in Q_1)(\exists q_2 \in Q_2)[q_1 \sim q_2] \wedge (\forall q_2 \in Q_2)(\exists q_1 \in Q_1)[q_1 \sim q_2] \Rightarrow (\Omega_1 \sim \Omega_2).$$

## 2.7 Мінімізація кінцевих автоматів.

Нехай $\Omega = \langle A, B, Q, q_o, \delta, \lambda \rangle$ - ініціальний кінцевий автомат. У загальному випадку КА може мати стани, в котрих не існують шляхи з

початкового стану (якщо скористатися термінологією теорії графів і згадати діаграми Мура для КА). Якщо КА починає свою роботу з початкового стану $q_o$, то в такі стани він потрапити не може. Такі стани КА називаються недосяжними, решта стану КА називаються досяжними. Очевидно, що видалення з КА недосяжних станів і переходів з них не впливає на роботу КА.

Формально ці визначення можна сформулювати наступним чином.

Стан $q \in Q$ називається досяжним тоді і лише тоді, коли існує вхідне слово, обробку якого з початкового стану кінцевий автомат закінчує в стані $q$, тобто:

$(q \in Q)$ – досяжне $<\overset{def}{=}>$ $(\exists\, w \in A^*)[\Delta(q_o, w) = q]$.

В іншому випадку стан q∈Q називається недосяжним, тобто:

$(q \in Q)$ – недосяжне $<\overset{def}{=}>$ $(\forall\, w \in A^*)[\Delta(q_o, w) \neq q]$.

Множина досяжних станів кінцевого автомата Ω будується за допомогою алгоритму, заснованого на індукції. На і-тому кроці будемо будувати множину станів, досяжних з початкового стану автомата під дією деякого вхідного слова. Довжина якого не перевершує і. Індуктивне визначення множини досяжних станів кінцевого автомата $\Omega = \langle A, B, Q, q_o, \delta, \lambda \rangle$:

$$\begin{cases} \tilde{Q}_0 = \{q_0\}; \\ \tilde{Q}_{i+1} = \tilde{Q}_i \cup \left( \bigcup\limits_{q \in Q_i \wedge a \in A} \delta(q, a) \right). \end{cases}$$

Очевидно, що безліч станів $\tilde{Q}_{|Q|}$ буде включати всі досяжні стани автомата Ω.

Кінцевий автомат називається мінімальним, якщо не існує еквівалентного йому автомата з меншою кількістю станів.

Мінімальний автомат $\overline{\Omega}$, еквівалентний даним автомата Ω, називається мінімальною формою автомата Ω.

Теорема 2.7.1 (про існування мінімальної форми автомата).

Для всякого кінцевого автомата $\Omega = \langle A, B, Q, q_o, \delta, \lambda \rangle$ існує мінімальна форма цього автомата.

Доведення (конструктивне).

Вкажемо алгоритм, за допомогою якого для всякого кінцевого автомата $\Omega = \langle A, B, Q, q_o, \delta, \lambda \rangle$ може бути побудована мінімальна форма цього автомата. Вхідні та вихідні алфавіти вихідного автомата і його мінімальної форми, очевидно, однакові відповідно. В якості множини $\overline{Q}$ станів автомата $\overline{\Omega}$ візьмемо $\pi$-розбиття автомата $\Omega$. При цьому кожен клас еквівалентності $K_i$ $\pi$-розбиття отождествім з деяким станом автомата $\overline{\Omega}$. Надалі, для простоти викладок, клас еквівалентності стану $q_i \in Q$ будемо позначати $\overline{q_i} \in \overline{Q}$. Ясно, що різні стану $q_i$ і $q_j$ можуть використовуватися для позначення одного і того самого стану автомата $\overline{\Omega}$, якщо вони належать одному класу еквівалентності $\pi$-розбиття автомата $\Omega$. В цьому випадку $\overline{q_i} = \overline{q_j}$.

Початковим станом $\overline{q_0}$ автомата $\overline{\Omega}$ оголосимо той клас еквівалентності $K_0$ $\pi$-розбиття, якому належить початковий стан $q_o$ вихідного автомата $\Omega$. Таким чином, в подальшому $\overline{q_i}$ - це, з одного боку, клас еквівалентності, з іншого боку, стан автомата.

Визначимо тепер функції переходів $\overline{\delta}: \overline{Q} \to \overline{Q}$ і виходів $\overline{\lambda}: \overline{Q} \to B$ автомата наступним чином:

$$(\forall \overline{q_i}, \overline{q_j} \in \overline{Q})(\forall a \in A)[\overline{\delta}(\overline{q_i}, a) = \overline{q_j} \Leftrightarrow \{\delta(q_i, a) = q_j\}]$$

$$(\forall \overline{q_i} \in \overline{Q})(\forall a \in A)(\forall b \in B)[\overline{\lambda}(\overline{q_i}, a) = b \Leftrightarrow \{\lambda(q_i, a) = b\}].$$

Таким чином, автомат $\overline{\Omega} = \langle A, B, \overline{Q}, \overline{q_0}, \overline{\lambda}, \overline{\delta} \rangle$ побудований. Доведемо його еквівалентність вихідного автомата $\Omega = \langle A, B, Q, q_o, \delta, \lambda \rangle$. Для цього доведемо, що $\forall q \in Q : q \sim \overline{q}$. Перш за все, мусимо переконатися, що для функцій $\Delta$ і $\overline{\Delta}$ виконується властивість: $(\forall q \in Q)(\forall w \in A^*)[\Delta(q, w) \in \overline{\Delta}(\overline{q}, w)]$.

Доказ проведемо індукцією по довжині вхідного слова *w*.

База індукції. Нехай $/ w /=0$. Тоді $w = \wedge$, $\Delta(q, \wedge) = q$ та, в силу визначення $\bar{\Delta}$, $\bar{\Delta}(\bar{q}, \wedge) = \bar{q}$. З визначення $\bar{q}$ зрозуміло, що $q \in \bar{q}$, тобто $\Delta(q, \wedge) \in \bar{q}$ та база індукції перевірена.

Припущення індукції. Нехай ствердження

$$(\forall q \in Q)(\forall w \in A^*)[\Delta(q, w) \in \bar{\Delta}(\bar{q}, w)]$$

Має місце для усіх вхідних слів *w*, таких, що $/ w /= n$.

Крок індукції. Нехай $w \in A^*$, та $/ w /= n+1$. Тоді вхідне слово $w$ можна представити у вигляді $w = w'a$, где $w' \in A^*$, $a \in A$, $| w' | = n$. Маємо:

$\Delta(q, w) = \Delta(q, w'a) = \delta(\Delta(q, w'), a)$ та $\bar{\Delta}(\bar{q}, w) = \bar{\Delta}(\bar{q}, w'a) = \bar{\delta}(\bar{\Delta}(\bar{q}, w'), a)$

Оскільки $/ w' /= n$, то за припущенням індукції $\Delta(q, w') \in \bar{\Delta}(\bar{q}, w')$. З візначення $\bar{\delta}$ випливає, що $\delta(\Delta(q, w'), a) \in \bar{\delta}(\bar{\Delta}(\bar{q}, w'), a)$, тобто $\Delta(q, w) \in \bar{\Delta}(\bar{q}, w)$ і тим самим, крок індукції доведений.

Отже, властивість $(\forall q \in Q)(\forall w \in A^*)[\Delta(q, w) \in \bar{\Delta}(\bar{q}, w)]$ доведено.

Доведемо тепер індукцією по довжині вхідного слова $w \in A^*$, що $\forall q \in Q : q \sim \bar{q}$, тобто $(\forall w \in A^*)[\Omega(q, w) = \bar{\Omega}(\bar{q}, w)]$

База індукції. Нехай $/ w /=0$. Тоді $w = \wedge$ та

$$\left.\begin{array}{l} \Omega(q, \wedge) = \wedge \\ \bar{\Omega}(\bar{q}, \wedge) = \wedge \end{array}\right\} \Rightarrow \Omega(q, \wedge) = \bar{\Omega}(\bar{q}, \wedge)$$

.

Припущення індукції. Нехай затвердження $(\forall w \in A^*)[\Omega(q, w) = \bar{\Omega}(\bar{q}, w)]$ має місце для всіх вхідних слів *w*, таких, що $/ w /\leq n$.

Крок індукції. Нехай $w \in A^*$ та $/ w /= n+1$. Тоді вхідне слово w можна представити у вигляді $w = w'a$, де $w' \in A^*, a \in A, / w' /= n$. Маємо:

$$\Omega(q, w) = \Omega(q, w'a) = \Omega(q, w')\lambda(\Delta(q, w'), a),$$
$$\bar{\Omega}(\bar{q}, w) = \bar{\Omega}(\bar{q}, w')\bar{\lambda}(\bar{\Delta}(\bar{q}, w'), a).$$

За припущенням індукції $\Omega(q, w') = \bar{\Omega}(\bar{q}, w')$. По доведеному раніше

$\Delta(q, w') \in \overline{\Delta}(\overline{q}, w')$ і тоді, в силу визначення $\overline{\lambda}$, має місце

$\lambda(\Delta(q, w'), a) = \overline{\lambda}(\overline{\Delta}(\overline{q}, w'), a).$

Отже, $\Omega(q, w) = \overline{\Omega}(\overline{q}, w)$ і, тим самим крок індукції доведений. Значить, доведено, що $\forall q \in Q : q \sim \overline{q}$.

Отримали, що для всякого стану $q$ автомата $\Omega$ існує еквівалентний йому стан $\overline{q}$ автомата $\overline{\Omega}$ і для кожного стану $\overline{q}$ автомата $\overline{\Omega}$ існує еквівалентний йому стан q автомата Ω (тобто за властивості розбиття, класи еквівалентності не порожні).

Доведемо, що автомат $\overline{\Omega} = \langle A, B, \overline{Q}, \overline{q_0}, \overline{\lambda}, \overline{\delta} \rangle$ - мінімальний. Якщо це не так, то існує еквівалентний йому автомат $\overline{\overline{\Omega}}$ з меншою кількістю станів. Одже, два стана автомата $\overline{\Omega}$ еквівалентні одному стану автомата $\overline{\overline{\Omega}}$ і тоді, в силу транзитивності відносини еквівалентності, ці стани еквівалентні між собою. Отримали, що статків, що належать різним класам еквівалентності, еквівалентні між собою, що неможливо. Отже, припущення про те, що автомат $\overline{\Omega}$ не є мінімальним, невірно і все доведено.

# 3 ВИБІР ЗАСОБІВ РОЗРОБКИ

Visual Studio Community 2017 – безкоштовне, повнофункціональне та розширюване інтегроване середовище розробки для створення сучасних додатків для Windows, Android та iOS, а також веб-додатків та хмарних сервісів. Функціонал даного середовища повністю покриває потреби, висунуті до середовища розробки нашого проекту. Вона є безкоштовною, що є найважливішим критерієм при виборі. Окрім цього вона слугує для створення додатків для Windows, що ідеально для нас підходить. Також Visual Studio підтримує можливість створення додатків Windows Forms. Тобто в ній ми можемо створювати додатки з формами та власним інтерфейсом, що значно розширює наші можливості при розробці проекту, на відмінну від консольних додатків.

C++ - об’єктно-орієнтована мова програмування. Для нас це дуже важливо, оскільки архітектура нашого проекту буде досить складною для того, щоб реалізувати увесь його функціонал в одному модулі. Тому було вирішено використовувати класи та інші риси об’єктно-орієнтованого програмування. Крім того, в C++ присутні методи для роботи зі списками, словниками, множинами та послідовностями, які будуть використовуватись в нашому проекті. Використання цих методів інтуїтивно зрозуміле, а їх реалізація дозволяє не приділяти зайвої уваги тому, скільки часу піде на виконання тої чи іншої функції.

# 4 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТ

Для реалізації продукту була вибрана мова програмування С++. У програмі створено клас котрий імітує працю автомата і реалізує всі алгоритми необхідні для вирішення завдання. Повний текст програми буде наведено у ДОДАТКУ А. Починається імплементування з визначення кінцевого автомату в якому:

- S – список станів кінцевого автомата;
- T – вхідна абетка;
- delta – функція переходу;
- lambda – функція виходу.

```
class DFA {
public:
        unsigned int sizeS;
        string *S;
        unsigned int sizeT;
        string *T;
        string **delta;
        bool **lambda;
```

Конструктор-обчислювач декартового добутку двох автоматів має такі властивості:

- обчислює добуток внутрішніх станів двох автоматів та виводить їх впорядковані пари;
- обчислює добуток функцій виходів двох автоматів та виводить диз’юнкцію значень.

Конструктор має такий вигляд:

```
DFA(DFA DFA1, DFA DFA2) {
    this->sizeT = DFA1.sizeT;
```

```
            this->sizeS = DFA1.sizeS * DFA2.sizeS;
            this->T = DFA1.T;
            this->S = new string[this->sizeS];
            this->delta = new string *[this->sizeS];
            for (unsigned int i = 0; i < this->sizeS; ++i)
                 this->delta[i] = new string[this->sizeT];
            this->lambda = new bool *[this->sizeS];
            for (unsigned int i = 0; i < this->sizeS; ++i)
                 this->lambda[i] = new bool[this->sizeT];
            for (unsigned int i = 0; i < DFA1.sizeS; ++i)
                 for (unsigned int j = 0; j < DFA2.sizeS; ++j) {
                      this->S[i * DFA2.sizeS + j] = "<" + DFA1.S[i]
+ "," + DFA2.S[j] + ">";
                      for (unsigned int k = 0; k < this->sizeT;
k++) {
                           this->delta[i * DFA2.sizeS + j][k] = "<"
+ DFA1.delta[i][k] + "," + DFA2.delta[j][k] + ">";
                           this->lambda[i * DFA2.sizeS + j][k] =
DFA1.lambda[i][k] || DFA2.lambda[j][k];
                      }
                 }
       }
```

Метод Minimize реалізує всі алгоритми необхідні для мінімізації кінцевого автомату отриманого в результаті декартового добутку. А саме:

- пошук недосяжних станів, тобто тих станів автомата, в які не існує жодного шляху з початкового стану в графі станів і переходів, рекурсивно відзначаємо досягнуті, виводимо результат;
- пошук еквівалентних станів. Перебір починається з першого стану (в програмі індекс 0). Порівнюється два стану та їх вихідні значення при першому символі вхідного алфавіту, якщо вони однакові то записуються в клас, якщо ні, створюємо новий клас;
- перетворюємо отримані класи в нові стани та будуємо мінімальний кінцевий автомат.

```
     void Minimize() {
          unsigned int **uIntDelta = DeltaToInt();
          vector<vector<unsigned int>> classes;
          vector<unsigned int> temp;
          bool addNew;
          bool *visited = new bool[sizeS];
          for (unsigned int i = 0; i < sizeS; i++)
```

```
     visited[i] = false;
MarkReachable(visited, 0);
cout << "\n";
for (unsigned int i = 0; i < sizeS; i++) {
     if (visited[i])
          cout << "State " << S[i] << " is visited\n";
     else
          cout << "State " << S[i] << " is not
          visited\n";
}
cout << "\n";
unsigned int countNotVisited = 0;
for (unsigned int i = 0; i < sizeS; i++)
     if (!visited[i])
          countNotVisited++;
string *newS = new string[sizeS - countNotVisited];
string **newDelta = new string *[sizeS -
countNotVisited];
bool **newLambda = new bool *[sizeS - countNotVisited];
unsigned int count = 0;
for (unsigned int i = 0; i < sizeS; i++)
     if (visited[i]) {
          newS[count] = S[i];
          newDelta[count] = delta[i];
          newLambda[count] = lambda[i];
          count++;
     }
sizeS -= countNotVisited;
S = newS;
delta = newDelta;
lambda = newLambda;
unsigned int *uIntOut = LambdaToUInt();
for (unsigned int i = 0; i < sizeS; i++)
     temp = { 0 };
classes.push_back(temp);
for (unsigned int i = 1; i < sizeS; i++) {
     addNew = true;
     for (unsigned int j = 0; j < classes.size(); j++)
     {
          if (uIntOut[i] == uIntOut[classes[j][0]]) {
               classes[j].push_back(i);
               addNew = false;
          }
     }
     if (addNew == true) {
          temp = { i };
          classes.push_back(temp);
     }
}
delete[] uIntDelta;
uIntDelta = DeltaToInt();
bool nothingChanged;
for (unsigned int k = 0; k < classes.size(); k++) {
```

```
restart:
     nothingChanged = true;
     if (classes[k].size() > 1)
          for (unsigned int i = 0; i <
     classes[k].size(); i++)
               for (unsigned int j = 0; j <
          classes[k].size(); j++)
                    if (i != j)
     if (!membersMatch(classes, uIntDelta, k, i, j))
     {
          vector<unsigned int> toAdd =
     PickIntoNewClass(classes, uIntDelta, k, j);
          ExpandVector(classes, toAdd, k);
          nothingChanged = false;
          goto restart;
     }
          if (!nothingChanged)
          k = 0;
}
cout << "Classes to minimize:\n";
for (unsigned int i = 0; i < classes.size(); i++) {
     cout << "class " << i << " : ";
     for (unsigned int j = 0; j < classes[i].size();
     j++)
          cout << S[classes[i][j]] << "  ";
     cout << "\n";
}
newLambda = new bool *[sizeS];
unsigned int **newUIntDelta = new unsigned int
*[sizeS];

newDelta = new string *[sizeS];
for (unsigned int i = 0; i < sizeS; i++)
     newDelta[i] = new string[sizeT];
for (unsigned int i = 0; i < sizeS; i++) {
auto tmp = GetTargetClasses(classes, uIntDelta, i);
     newUIntDelta[i] = tmp;
     newLambda[i] = lambda[classes[i][0]];
}
delete[] S;
S = new string[sizeS];
for (unsigned int i = 0; i < sizeS; i++)
     S[i] = (char)(65 + i);
for (unsigned int i = 0; i < sizeS; i++)
     for (unsigned int j = 0; j < sizeT; j++)
          newDelta[i][j] = S[newUIntDelta[i][j]];
delete[] lambda;
delete[] delta;
delta = newDelta;
lambda = newLambda;
for (unsigned int i = 0; i < sizeS; i++)
     for (unsigned int j = 0; j < sizeT; j++) {
          delta[i][j] = newDelta[i][j];
```

```
                    lambda[i][j] = newLambda[i][j]; }
}
```

Було вирішено, що всі необхідні для обчислення дані буде введено у вхідних файлах. Для цього створено 2 файли «DFAs.txt» та «input.txt». У головній функції зчитується вхідний алфавіт та розмір кінцевих автоматів з файлу «DFAs.txt» та вхідне слово з файлу «input.txt»

```
int main(int argc, char *argv[]) {
      ifstream input_DFAs("DFAs.txt");
      unsigned int sizeS1, sizeS2, sizeT;
      input_DFAs >> sizeS1 >> sizeS2 >> sizeT;
      string *T = new string[sizeT];
      for (unsigned int i = 0; i < sizeT; ++i)
            input_DFAs >> T[i];
      DFA DFA1(sizeS1, sizeT, T, input_DFAs);
      DFA DFA2(sizeS2, sizeT, T, input_DFAs);
      ifstream input_data("input.txt");
      string input, output;
      getline(input_data, input);
```

Для зчитування автоматів створенний конструктор, котрий виділяє пам’ять для нового автомату и записує дані у поля класу.

```
DFA(unsigned int sizeS, unsigned int sizeT, string *T,
ifstream &in) {
     this->sizeT = sizeT;
     this->sizeS = sizeS;
     this->T = T;
     S = new string[sizeS];
     this->delta = new string *[sizeS];
     for (unsigned int i = 0; i < sizeS; ++i)
          this->delta[i] = new string[sizeT];
     this->lambda = new bool *[sizeS];
     for (unsigned int i = 0; i < sizeS; ++i)
          this->lambda[i] = new bool[sizeT];
     for (unsigned int i = 0; i < sizeS; i++) {
          in >> this->S[i];
          for (unsigned int j = 0; j < sizeT; j++)
               in >> this->delta[i][j];
          for (unsigned int j = 0; j < sizeT; j++)
               in >> this->lambda[i][j];
     }
 }
```

Для обробки вхідної строки і друку вихідної створений метод, який входить в автомат з початкового стану, посимвольно перебирає вхідну строку і якщо вхідний символ збігається з функцією переходу в поточному стані, то записується у вивод.

```
string Process(string input) {
     string output = "";
     unsigned int length = input.length();
     if (length > 0) {
          unsigned int current_index = 0;
          for (unsigned int i = 0; i < length; i++)
          for (unsigned int j = 0; j < sizeT; j++)
               if (input[i] == T[j][0]) {
          output += to_string(lambda[current_index][j]);
     current_index = this->GetNewIndex(current_index, j);
          break;
          }
     }
     return output;
  }
```

Метод перекладу значень функції виходу в двійкову форму, а потім її в дестякову має такий вигляд:

```
unsigned int *LambdaToUInt() {
     unsigned int *out = new unsigned int[sizeS];
     for (unsigned int i = 0; i < sizeS; i++) {
          unsigned int sum = 0;
          for (unsigned int j = 0; j < sizeT; j++)
               sum += lambda[i][j] * pow(2, sizeT - j - 1);
          out[i] = sum;
     }
     return out;
  }
```

Виводимо кінцевий автомат, а також вхідну та вихідну строку:

```
     cout << "First DFA: \n";
     for (unsigned int i = 0; i < sizeS1; i++) {
          cout << DFA1.S[i] << " |";
          for (unsigned int j = 0; j < sizeT; j++)
               cout << " " << DFA1.delta[i][j] << " ";
```

```
        cout << " | ";
        for (unsigned int j = 0; j < sizeT; j++)
             cout << DFA1.lambda[i][j] << "   ";
        cout << "\n";
}
output = DFA1.Process(input);
cout << "\nInput line: " << input << "\nOutput line: "
 << output << "\n";
```

# ВИСНОВОК

У результаті виконання цього проекту був виконаний аналіз алгоритмів побудування кінцевих алгоритмів. Також була побудована програма, котра складається з двох частин. Перша частина будь-яким двом кінцевим автоматам-акцепторам будує автомат, котрий розпізнає множину слів першого автомата в об’єднанні з множиною слів другого автомата. Друга частина програми мінімізує ініціальний кінцевий автомат, котрий заданий таблично.

# СПИСОК ЛІТЕРАТУРИ

1. Джон Хопкрофт, Раджив Мотвани, Джеффри Ульман. Введение в теорию автоматов, языков и вычислений. 2-е изд.. : М.: Издательский дом «Вильямс», 2002. -528 с. : ISBN 5-8459-0261-4 (рус.)
2. В.М. Глушков. Синтез цифровых автоматов. / В.М.Глушков. . - 3-е изд. – СПб.: Питер, М.: Физматгиа, 1962. -476 с.
3. Теория автоматов / Ю.Г.Карпов . - 2-е изд. – СПб.: Питер, 2003. -208с.
4. Елементи теорії дискретних систем. / В. К. Булитко. – Одеса, 1997. -76с.
5. Минский М. Вычисления и автоматы. / Минский М. М., Мир, 1971. – 366с.

# ДОБУТОК А

# ТЕКСТ ПРОГРАМИ

```
class DFA {
public:
     unsigned int sizeS;
     string *S;
     unsigned int sizeT;
     string *T;
     string **delta;
     bool **lambda;
     DFA(unsigned int sizeS, unsigned int sizeT, string *T,
ifstream &in) {
          this->sizeT = sizeT;
          this->sizeS = sizeS;
          this->T = T;
          S = new string[sizeS];
          this->delta = new string *[sizeS];
          for (unsigned int i = 0; i < sizeS; ++i)
               this->delta[i] = new string[sizeT];
          this->lambda = new bool *[sizeS];
          for (unsigned int i = 0; i < sizeS; ++i)
               this->lambda[i] = new bool[sizeT];
          for (unsigned int i = 0; i < sizeS; i++) {
               in >> this->S[i];
               for (unsigned int j = 0; j < sizeT; j++)
                    in >> this->delta[i][j];
               for (unsigned int j = 0; j < sizeT; j++)
                    in >> this->lambda[i][j];
          }
     }
     DFA(DFA DFA1, DFA DFA2) {
          this->sizeT = DFA1.sizeT;
          this->sizeS = DFA1.sizeS * DFA2.sizeS;
```

```
        this->T = DFA1.T;
        this->S = new string[this->sizeS];
        this->delta = new string *[this->sizeS];
        for (unsigned int i = 0; i < this->sizeS; ++i)
            this->delta[i] = new string[this->sizeT];
        this->lambda = new bool *[this->sizeS];
        for (unsigned int i = 0; i < this->sizeS; ++i)
            this->lambda[i] = new bool[this->sizeT];
        for (unsigned int i = 0; i < DFA1.sizeS; ++i)
            for (unsigned int j = 0; j < DFA2.sizeS; ++j) {
            this->S[i * DFA2.sizeS + j] = "<" + DFA1.S[i] +
            "," + DFA2.S[j] + ">";
            for (unsigned int k = 0; k < this->sizeT; k++) {
                this->delta[i * DFA2.sizeS + j][k] = "<" +
             DFA1.delta[i][k] + "," + DFA2.delta[j][k] + ">";
                this->lambda[i * DFA2.sizeS + j][k] =
             DFA1.lambda[i][k] || DFA2.lambda[j][k];
                }
            }
    }
    string Process(string input) {
        string output = "";
        unsigned int length = input.length();
        if (length > 0) {
            unsigned int current_index = 0;
            for (unsigned int i = 0; i < length; i++)
                for (unsigned int j = 0; j < sizeT; j++)
                    if (input[i] == T[j][0]) {
                        output +=
                    to_string(lambda[current_index][j]);
                        current_index = this-
                    >GetNewIndex(current_index, j);
                        break;
                    }
        }
```

```
        return output;
    }
    void Minimize() {
        unsigned int **uIntDelta = DeltaToInt();
        vector<vector<unsigned int>> classes;
        vector<unsigned int> temp;
        bool addNew;
        bool *visited = new bool[sizeS];
        for (unsigned int i = 0; i < sizeS; i++)
            visited[i] = false;
        MarkReachable(visited, 0);
        cout << "\n";
        for (unsigned int i = 0; i < sizeS; i++) {
            if (visited[i])
                cout << "State " << S[i] << " is visited\n";
            else
            cout << "State " << S[i] << " is not visited\n";
        }
        cout << "\n";
        unsigned int countNotVisited = 0;
        for (unsigned int i = 0; i < sizeS; i++)
            if (!visited[i])
                countNotVisited++;
        string *newS = new string[sizeS - countNotVisited];
        string **newDelta = new string *[sizeS -
countNotVisited];
        bool **newLambda = new bool *[sizeS - countNotVisited];
        unsigned int count = 0;
        for (unsigned int i = 0; i < sizeS; i++)
            if (visited[i]) {
                newS[count] = S[i];
                newDelta[count] = delta[i];
                newLambda[count] = lambda[i];
                count++;
```

```
     }
sizeS -= countNotVisited;
S = newS;
delta = newDelta;
lambda = newLambda;
unsigned int *uIntOut = LambdaToUInt();
for (unsigned int i = 0; i < sizeS; i++)
     temp = { 0 };
classes.push_back(temp);
for (unsigned int i = 1; i < sizeS; i++) {
     addNew = true;
     for (unsigned int j = 0; j < classes.size(); j++)
      {
          if (uIntOut[i] == uIntOut[classes[j][0]]) {
               classes[j].push_back(i);
               addNew = false;
          }
     }
     if (addNew == true) {
          temp = { i };
          classes.push_back(temp);
     }
}
delete[] uIntDelta;
uIntDelta = DeltaToInt();
bool nothingChanged;
for (unsigned int k = 0; k < classes.size(); k++) {
restart:
     nothingChanged = true;
     if (classes[k].size() > 1)
for (unsigned int i = 0; i < classes[k].size(); i++)
for (unsigned int j = 0; j < classes[k].size(); j++)
                    if (i != j)
     if (!membersMatch(classes, uIntDelta, k, i, j)) {
```

```
vector<unsigned int> toAdd = PickIntoNewClass(classes,
 uIntDelta, k, j);
          ExpandVector(classes, toAdd, k);
               nothingChanged = false;
               goto restart;
          }
          if (!nothingChanged)
          k = 0;
}
cout << "Classes to minimize:\n";
for (unsigned int i = 0; i < classes.size(); i++) {
     cout << "class " << i << " : ";
for (unsigned int j = 0; j < classes[i].size(); j++)
     cout << S[classes[i][j]] << "  ";
     cout << "\n";
}
newLambda = new bool *[sizeS];
unsigned int **newUIntDelta = new unsigned int
*[sizeS];
newDelta = new string *[sizeS];
for (unsigned int i = 0; i < sizeS; i++)
     newDelta[i] = new string[sizeT];
for (unsigned int i = 0; i < sizeS; i++) {
auto tmp = GetTargetClasses(classes, uIntDelta, i);
     newUIntDelta[i] = tmp;
     newLambda[i] = lambda[classes[i][0]];
}
delete[] S;
S = new string[sizeS];
for (unsigned int i = 0; i < sizeS; i++)
     S[i] = (char)(65 + i);
for (unsigned int i = 0; i < sizeS; i++)
     for (unsigned int j = 0; j < sizeT; j++)
          newDelta[i][j] = S[newUIntDelta[i][j]];
```

```
        delete[] lambda;
        delete[] delta;
        delta = newDelta;
        lambda = newLambda;
        for (unsigned int i = 0; i < sizeS; i++)
            for (unsigned int j = 0; j < sizeT; j++) {
                delta[i][j] = newDelta[i][j];
                lambda[i][j] = newLambda[i][j];
            }
    }
    void MarkReachable(bool *visited, unsigned int currentState)
    {
        visited[currentState] = true;
        unsigned int target1, target2;
        target1 = GetNewIndex(currentState, 0);
        target2 = GetNewIndex(currentState, 1);
        if (!visited[target1])
            MarkReachable(visited, target1);
        if (!visited[target2])
            MarkReachable(visited, target2);
    }
    private:
        unsigned int GetNewIndex(unsigned int delta_i, unsigned
    int delta_j) {
        for (unsigned int i = 0; i < this->sizeS; i++)
            if (this->delta[delta_i][delta_j] == this->S[i])
                return i;
        return 9999;
    }
    unsigned int *LambdaToUInt() {
        unsigned int *out = new unsigned int[sizeS];
        for (unsigned int i = 0; i < sizeS; i++) {
            unsigned int sum = 0;
            for (unsigned int j = 0; j < sizeT; j++)
```

```
                    sum += lambda[i][j] * pow(2, sizeT - j - 1);
               out[i] = sum;
          }
          return out;
     }
     unsigned int **DeltaToInt() {
          unsigned int ** out = new unsigned int *[sizeS];
          for (unsigned int i = 0; i < sizeS; i++)
               out[i] = new unsigned int[sizeT];
          for (unsigned int i = 0; i < sizeS; i++)
               for (unsigned int j = 0; j < sizeT; j++)
                    for (unsigned int k = 0; k < sizeS; k++)
                         if (delta[i][j] == S[k]) {
                              out[i][j] = k;
                              break;
                         }
          return out;
     }
     bool membersMatch(vector<vector<unsigned int>> classes,
unsigned int **uIntDelta, unsigned int currentClass, unsigned
int firstToCompare, unsigned int secondToCompare) {
          unsigned int firstTarget, secondTarget;
          for (unsigned int n = 0; n < sizeT; n++) {
          for (unsigned int i = 0; i < classes.size(); i++){
          for (unsigned int j = 0; j < classes[i].size(); j++)
          {
               if (classes[i][j] ==
          uIntDelta[classes[currentClass][firstToCompare]][n])
               firstTarget = i;
               if (classes[i][j] ==
          uIntDelta[classes[currentClass][secondToCompare]][n])
               secondTarget = i;
               }
          }
               if (firstTarget != secondTarget)
```

```
                    return false;
          }
          return true;
     }
     vector<unsigned int> PickIntoNewClass(vector<vector<unsigned
int>> classes, unsigned int **uIntDelta, unsigned int
class_index, unsigned int member_index) {
          vector<unsigned int> outVector;
     outVector.push_back(classes[class_index][member_index]);
for (unsigned int i = 0; i < classes[class_index].size(); i++)
          if (i != member_index && membersMatch(classes,
uIntDelta, class_index, member_index, i))
          outVector.push_back(classes[class_index][i]);
          return outVector;
     }
     void ExpandVector(vector<vector<unsigned int>> &classes,
vector<unsigned int> newVector, unsigned int input_position) {
          vector<vector<unsigned int>> out;
          vector<unsigned int> temp, shrinked;
          for (unsigned int i = 0; i < classes.size(); i++) {
               if (i == input_position) {
          for (unsigned int j = 0; j < classes[i].size(); j++) {
                         bool moved = false;
          for (unsigned int k = 0; k < newVector.size(); k++)
          if (classes[i][j] == newVector[k]) {
                    moved = true;
                    break;
               }
               if (!moved)
               shrinked.push_back(classes[i][j]);
          }
               out.push_back(shrinked);
               out.push_back(newVector);
          }
               else {
```

```
                temp = classes[i];
                out.push_back(temp);
            }
            }
            classes.swap(out);
        }
        unsigned int *GetTargetClasses(vector<vector<unsigned int>>
classes, unsigned int **uIntDelta, unsigned int currentClass) {
            unsigned int *out = new unsigned int[sizeT];
            for (unsigned int n = 0; n < sizeT; n++)
            for (unsigned int i = 0; i < classes.size(); i++)
            for (unsigned int j = 0; j < classes[i].size(); j++)
        if (uIntDelta[classes[currentClass][0]][n] == classes[i][j])
                    out[n] = i;
            return out;
        }
};

int main(int argc, char *argv[]) {
        ifstream input_DFAs("DFAs.txt");
        unsigned int sizeS1, sizeS2, sizeT;
        input_DFAs >> sizeS1 >> sizeS2 >> sizeT;
        string *T = new string[sizeT];
        for (unsigned int i = 0; i < sizeT; ++i)
            input_DFAs >> T[i];
        DFA DFA1(sizeS1, sizeT, T, input_DFAs);
        DFA DFA2(sizeS2, sizeT, T, input_DFAs);
        cout << "First DFA: \n";
        for (unsigned int i = 0; i < sizeS1; i++) {
            cout << DFA1.S[i] << " |";
            for (unsigned int j = 0; j < sizeT; j++)
                cout << " " << DFA1.delta[i][j] << " ";
            cout << " | ";
            for (unsigned int j = 0; j < sizeT; j++)
```

```
                cout << DFA1.lambda[i][j] << "   ";
            cout << "\n";
        }
        ifstream input_data("input.txt");
        string input, output;
        getline(input_data, input);
        output = DFA1.Process(input);
        cout << "\nInput line: " << input << "\nOutput line: " <<
output << "\n";
        cout << "\nSecond DFA:\n";
        for (unsigned int i = 0; i < sizeS2; i++) {
            cout << DFA2.S[i] << " |";
            for (unsigned int j = 0; j < sizeT; j++)
                cout << " " << DFA2.delta[i][j] << " ";
            cout << " | ";
            for (unsigned int j = 0; j < sizeT; j++)
                cout << DFA2.lambda[i][j] << "   ";
            cout << "\n";
        }
        output = DFA2.Process(input);
        cout << "\nInput line: " << input << "\nOutput line: " <<
output << "\n";
        DFA newDFA(DFA1, DFA2);
        cout << "\nNew DFA:\n";
        for (unsigned int i = 0; i < newDFA.sizeS; i++) {
            cout << newDFA.S[i] << " |";
            for (unsigned int j = 0; j < sizeT; j++)
                cout << " " << newDFA.delta[i][j] << " ";
            cout << " | ";
            for (unsigned int j = 0; j < sizeT; j++)
                cout << newDFA.lambda[i][j] << "   ";
            cout << "\n";
        }
        output = newDFA.Process(input);
```

```
     cout << "\nAfter union: " << "\nInput line: " << input <<
"\nOutput line: " << output << "\n";
     newDFA.Minimize();
     cout << "\nMinimized new DFA:\n";
     for (unsigned int i = 0; i < newDFA.sizeS; i++) {
          cout << newDFA.S[i] << " |";
          for (unsigned int j = 0; j < sizeT; j++)
               cout << " " << newDFA.delta[i][j] << " ";
          cout << " | ";
          for (unsigned int j = 0; j < sizeT; j++)
               cout << newDFA.lambda[i][j] << "   ";
          cout << "\n";
     }
     output = newDFA.Process(input);
     cout << "\nAfter minimization:\nInput line: " << input <<
"\nOutput line: " << output << "\n\n";
     system("pause");
     return 0;
}
```