**ЗМІСТ**

[**ВСТУП 3**](#_Toc7970334)

[**1 ПОСТАНОВКА ІНДИВІДУАЛЬНОГО ЗАВДАННЯ 4**](#_Toc7970335)

[**2 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ 6**](#_Toc7970336)

[**3 ВИБІР ТЕХНОЛОГІЧНОГО РІШЕННЯ ЗАДАЧІ 8**](#_Toc7970337)

[**4 АЛГОРИТМ РІШЕННЯ ЗАДАЧІ 11**](#_Toc7970338)

[**5 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ 14**](#_Toc7970339)

[**ВИСНОВКИ 19**](#_Toc7970340)

[**ПЕРЕЛІК ПОСИЛАНЬ 20**](#_Toc7970341)

**ДОДАТОК А.** Лістинг програми

# ВСТУП

Дослідження в галузі теорії автоматів почалися ще в середині 50-х років минулого століття. Не дивлячись на свою простоту, модель скінченного автомата виявилася надзвичайно зручною у застосуванні не тільки в інформатиці, але і в багатьох інших галузях інженерної діяльності. Великий інтерес до цієї теорії пояснюється саме широкими можливостями її застосування.

На переддипломній практиці було розроблено програмне забезпечення, для перетворення регулярної граматики, яку задає користувач, до недетермінованого скінченного автомату.

Дане програмне забезпечення створено для вчителів, студентів, а також розробників в галузі системного програмування. Не маючи аналогів, програмне забезпечення набуде широкого розповсюдження серед фахівців які застосовують модель скінченного автомату. Головною ціллю програмного продукту є зменшення часу на виконання складного алгоритму перетворення регулярної граматики до скінченного автомату, та подальше його застосування у навчальних та наукових цілях.

Скінчено-автоматна модель використовується в автоматиці для написання алгоритмів функціонування широкого класу пристроїв і систем дискретної техніки. Скінчено-автоматний опис алгоритмів необхідний не тільки для пояснення алгоритмів на початковому етапі проектування, а й для систематичної процедури синтезу пристроїв і програм.

# 1 ПОСТАНОВКА ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

На переддипломній практиці було поставлено наступне завдання: створити програмне забезпечення яке дає можливість перетворити регулярну граматику, яку користувач задає власноруч, до скінченного автомату.

Після аналізу заданої теми, було поставлено наступні вимоги до розробки програмного продукту:

* оптимізувати програмний продукт, для збільшення швидкодії, швидкості на виконання певної дії, для користувачів з застарілим програмним або апаратним забезпеченням;
* створити програмний інтерфейс який буде зрозумілий для використання користувачам, які не розуміють принципи системного програмування, а також застосування до інтерфейсу правил ергономічності, гарної взаємодії людини та техніки;
* створення кросплатформенного програмного продукту на мові програмування Java, для запуску програми на операційних системах Windows, Linux;
* використання патерну проектування MVC для розділення коду програми на логічні структури;
* використання зручної середи програмування IntelIj IDEA для збільшення швидкості розробки програмного забезпечення на мові програмування Java;
* використання платформи JavaFX, для створення ергономічного та новітнього інтерфейсу;
* створення функцій програмного продукту для перегляду функцій переходу скінченного автомату, графу перетворення скінченного автомату, структуру скінченного автомату.

Основними вимогами для програмного та апаратного забезпечення користувача, для забезпечення ефективності програмного продукту, є:

* ліцензійна операційна система Windows або Linux;
* встановлений Java Development Kit (JDK) версії 8 та вище, для запуску додатків на мові програмування Java;
* графічна карта з відео пам’яттю 512 мегабайт та більше, для створення графу скінченного автомату;

Також однією з основних вимог до програмного забезпечення є створення показового прикладу, як приклад було вибрано скінченний автомат механічного годинника, рисунок 1.1.



Рисунок 1.1 – Ілюстрація прикладу

# 2 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

При виконанні пошуку аналогів або прототипів програм за заданою темою, не було знайдено програмних продуктів або веб-сайтів для перетворення регулярної граматики до скінченного автомату. Отже розробка даного програмного продукту є дуже актуальною, необхідною для розробників та студентів у галузі системного програмування.

Все різноманіття видів цифрових машин можна віднести до одного класу кінцевих автоматів. Це означає, що вони володіють загальними властивостями:

* по-перше, будь-яка цифрова обчислювальна машина складається з кінцевої безлічі елементів, кожен з яких в будь-який даний момент часу може знаходитися лише в одному з кінцевого числа стійких станів. Тому і вся машина має лише кінцеву безліч стійких станів;
* по-друге, кожна цифрова машина працює послідовно, її операції синхронізовані сигналами ретельно налаштованого електронного годинника (зазвичай від 106 до 109 сигналів в секунду). Відповідно до цього стану машини змінюються в чіткій послідовності;
* по-третє, цифрова обчислювальна машина є детермінованим пристроєм: при наявності повної інформації про внутрішніх станах всіх елементів машини і всіх її входів наступний стан машини визначено однозначно.
* без перебільшення можна сказати, що теорія автоматів є одним з фундаментальних блоків сучасної теоретичної і практичної інформатики. Разом з класичними додатками теорії автоматів, такими як проектування вбудованих систем логічного управління, обробка текстів і побудова компіляторів штучних мов, останнім часом з'явилися нові, нетрадиційні області застосування цієї теорії – специфікація і верифікація систем взаємодіючих процесів (зокрема, протоколів комунікації), мови опису документів і об'єктно-орієнтованих програмних систем, оптимізація, логічних програм.

Кінцеві автомати значно розширюють можливості управління виконанням фонових завдань. Їх використання робить можливим надання фоновими потоками інформації про стан виконання, а також звернення інших потоків із запитами до фонового потоку на виконання певних дій, наприклад, із запитом на припинення виконання фонової роботи.

Додатки найкраще проектувати на основі одно поточному моделі, залучаючи фонову обробку лише тоді, коли вона необхідна.

Кінцеві автомати вельми зручно застосовувати в іграх, в яких персонажі, що переміщаються в області гри, можуть діяти в різних режимах (наприклад, персонаж біжить вперед або назад, підіймається сходами, падає та інше).

Використання автоматів з чітко визначеними правилами для кожного персонажа, які задають варіанти його поведінки в різних станах і допустимі переходи з одного стану в інший, дозволяє створювати складні варіанти поведінки як персонажів, керованих комп'ютером, так і персонажів, керованих користувачем.

Ідея застосування кінцевих автоматів є надзвичайно корисною концепцією, плідність якої пройшла перевірку часом. Використання кінцевих автоматів дозволяє розробникам створювати добре організовані додатки з гнучкими можливостями. Їх застосування дозволяє створювати ясний, зрозумілий і надійно функціонує код.

Намагайтеся уникати неявного управління станами програми, але не шкодуйте часу на розробку явно сформованої моделі станів.

Використання кінцевих автоматів стало вже звичайною при проектуванні додатків для настільних комп'ютерів, серверів і мобільних пристроїв.

Переваги, які забезпечує застосуванням кінцевих автоматів, найпомітніше проявляються в разі додатків для мобільних пристроїв, що вимагають економного розподілу екранного простору, пам'яті, обчислювальної потужності і інших ресурсів.

Можна визначити основні переваги скінченного автомату:

* формалізація. При описі завдання через кінцевий автомат необхідно замислюватися про всі станах, в яких може опинитися об'єкт. Так отримуємо і документацію, і можемо виявити не розглянуті в постановці завдання моменти. Відповідно, спрощується тестування коду;
* контроль потоків даних. Явне управління послідовністю викликів (потоком управління);
* контроль помилок. Якщо кінцевий автомат потрапляє в помилкове стан, то це просто означає, що при проектуванні забули визначити ще один стан. Єдина точка входу для логування та збору статистики. Наприклад, SwiftyStateMachine дозволяє явно вказати конкретний блок, в якому можна залоговані, що відбувається з нашими даними. Це істотно спрощує налагодження додатків;
* історія операцій. Використовуючи кінцевий автомат, можна реалізувати скасування операцій. Або, навпаки, відновити всю картину переходів між станами. Стек операцій зазвичай зберігається в самому стані.

Отже, скінченні автомати варто застосовувати, коли дійсно потрібна формалізація, висока тестованість коду і потрібно розподілити завдання між розробниками.

# 3 ВИБІР ТЕХНОЛОГІЧНОГО РІШЕННЯ ЗАДАЧІ

Як основну мову програмування було вибрано мову програмування Java.

Java – це об’єктно–орієнтована мова програмування, розроблена Джеймсом Ослингом і запущена в 1995 році в якості одного із ключових компонентів платформи Java Sun Microsystems.

Cинтаксис мови взятий в більшості з мови програмування С++ і С, про те Java має більш простішу об’єктивну модель і більш низький рівень об’єктів. Java – додатки як правило трансльовані в байт–код (довжина кожного коду операції – один байт). Який може бути оброблений любою віртуальною машиною Java (Java Virtual Mashine. JVM) незалежно від комп’ютерної архітектури.

Першочергово реалізація посилань на компілятори Java, віртуальних машин і бібліотек класів, була представлена компанією Sun в 1995 році. В травні 1997 року разом з Java Communitz Process Sun розробила технологію Java згідно з GNU General Public License. Інші компанії також шукали альтернативи технології запропонованої Sun як GNU Compiler, для Java і GNU Classpath.

Java залишається стандартом, яким керується Java Community Process (процес який дозволяє заінтересованим лицям приймати участь у формуванні майбутніх специфікацій Java). Sun запропонувала більшість частин Java безкоштовно, незалежно на статус власника програмного забезпечення. Доходи від Java поступають за рахунок продажі спеціалізованих продуктів, таких як Enterprise Java System.

Розробляючи мову спеціалісти керувались п’ятьма завданнями, які були перетворені в наступні принципи:

* простота в використанні, об’єктна орієнтованість і легкість вивчення;
* надійність і безпечність;
* незалежність від архітектури;
* можливість інтерпретації;
* інтерактивність і динамічність.

Можливість Java виконувати свій код на любій із підтримуваних платформ, досягається за рахунок того що її програми транслюються в проміжне уявлення, яке називається байт кодом. Байт код може інтерпретуватись в любій системі, в якій присутнє середовище виконання Java.

Значна кількість ранніх систем, в якій вводилась можливість незалежності від платформи, зіштовхувалась з не достатком заключному в зниженні продуктивності.

Байт код легко переводиться безпосередньо в рідні машинні коди, незалежно на те що в Java використовується інтерпретатор при якому досягається дуже висока продуктивність. Інструкції java bytecode похожі з інструкціями машинного коду, але призначені для віртуальної машини і створені спеціально для апаратного власника.

Кінцеві користувачі застосовують як правило Java Runtime Environment (JRE конфігурація віртуальної машини, необхідна для виконання Java додатків) встановлену як правило на власному комп’ютері для додатків Java або в веб браузері для Java аплетів.

Віртуальна машина Java (Java Virtual Machine) це ключовий компонент платформи Java, представляючи собою віртуальне середовище, віртуальний комп’ютер з характеристиками нормального комп’ютера (реєстром, кодами, набором інформації) який керує додатками Java.

Віртуальна машина Java (Java Virtual Machine) інтерпретує і виконує байт код Java, передчасно створений із початкового тексту Java програми компілятором Java.

Бібліотеки в терміні мов програмування це ресурси програмного забезпечення, які потрібні мові для функціонування.  Java використовує такі бібліотеки, які зберігають спеціальні директорії, в папці додатку Java. Звідтам вони завантажуються кожен раз, коли ініціалізується додаток Java. Стандартні бібліотеки Java забезпечують загальний доступ до конкретних функцій Java, які реалізують основні можливості мови.

Для візуального представлення, а також розділення за структурою MVC було обрано технологію JavaFX.

JavaFX – платформа та набір інструментів для створення [насичених інтернет-застосунків](https://uk.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D0%B8%D1%87%D0%B5%D0%BD%D0%B8%D0%B9_%D1%96%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82-%D0%B7%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA) (Rich Internet Applications, RIA) з можливістю під вантаження медіа та змісту. Вперше продемонстровано [Sun Microsystems](https://uk.wikipedia.org/wiki/Sun_Microsystems) на Міжнародній конференції [Java](https://uk.wikipedia.org/wiki/Java)-розробників JavaOne у травні [2007](https://uk.wikipedia.org/wiki/2007). JavaFX включає в себе набір утиліт, за допомогою яких веб-розробники та дизайнери можуть швидко створювати та надавати розвинуті інтернет-[застосунки](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BA%D0%B8) для десктопів, мобільних пристроїв, телебачення та інших платформ.

JavaFX складається з JavaFX Script і JavaFX Mobile. Починаючи з випуску JavaFX 2.0 забезпечено можливість створення JavaFX-застосунків, написаних цілком мовою [Java](https://uk.wikipedia.org/wiki/Java). Для розробки [застосунків](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA) доступний багатий графічний і мультимедійний [API](https://uk.wikipedia.org/wiki/API), що спрощує створення візуальних програм.

Однією з найважливіших особливостей цієї платформи називається функція Drag-to-Install, яка дозволяє встановлювати JavaFX-застосунки простим перетягуванням з вікна [браузера](https://uk.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80) на [робочий стіл](https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B1%D0%BE%D1%87%D0%B8%D0%B9_%D1%81%D1%82%D1%96%D0%BB).

JavaFX 1.0 включив підтримку [відео](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%B4%D0%B5%D0%BE) (JavaFX Video Support) на базі кодеків On2 Technologies і три ключові компоненти: JavaFX Development Environment (компілятор і середовище виконання, графічні, медійні і веб-бібліотеки, інтегроване середовища розробки [NetBeans](https://uk.wikipedia.org/wiki/NetBeans) IDE 6.5, мобільний емулятор), JavaFX Production Suite (набір інструментів і [плагінів](https://uk.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%B3%D1%96%D0%BD) для імпорту графічних об'єктів, створених в дизайнерських пакетах, таких як [Adobe Photoshop](https://uk.wikipedia.org/wiki/Adobe_Photoshop) і [Adobe Illustrator](https://uk.wikipedia.org/wiki/Adobe_Illustrator)), JavaFX Desktop (експлуатує функції Java SE 6 Update 10, включаючи вже згадану Drag-to-Install).

На момент випуску JavaFX 1.0 доступна тільки для [ОС](https://uk.wikipedia.org/wiki/%D0%9E%D0%A1) [Windows](https://uk.wikipedia.org/wiki/Windows) і [Mac OS](https://uk.wikipedia.org/wiki/Mac_OS), але розробники обіцяють незабаром забезпечити підтримку й інших платформ. Sun дещо спізнюється на ринку потокового медіа: якщо у 1995 році [Java](https://uk.wikipedia.org/wiki/Java) була одноосібним лідером і законодавцем, то сьогодні її серйозно підтискають конкуренти, такі як [Microsoft Silverlight](https://uk.wikipedia.org/wiki/Microsoft_Silverlight), [Adobe Flash](https://uk.wikipedia.org/wiki/Adobe_Flash) і [AIR](https://uk.wikipedia.org/wiki/Adobe_AIR). Щоб підвищити інтерес до своєї платформи, Sun також планує розкрити сирцеві коди JavaFX.

Головним нововведенням JavaFX 2.0, що вийшов вже під орудою [Oracle](https://uk.wikipedia.org/wiki/Oracle_Corporation) у жовтні 2011, стала відмова від використання спеціальної [скриптової мови](https://uk.wikipedia.org/wiki/%D0%A1%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B2%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) JavaFX Script на користь стандартного [Java](https://uk.wikipedia.org/wiki/Java) [API](https://uk.wikipedia.org/wiki/API), що дозволяє створювати JavaFX застосунки мовами, що базуються на [JVM](https://uk.wikipedia.org/wiki/JVM) — [JRuby](https://uk.wikipedia.org/wiki/JRuby), [Clojure](https://uk.wikipedia.org/wiki/Clojure), [Scala](https://uk.wikipedia.org/wiki/Scala) і [Groovy](https://uk.wikipedia.org/wiki/Groovy). Для розробки застосунків доступний багатий графічний і мультимедійний API, що спрощує створення візуальних програм.

Разом з випуском Oracle підтвердив перехід проекту на відкриту модель розробки.[[2]](https://uk.wikipedia.org/wiki/JavaFX#cite_note-2) У найближчому майбутньому Oracle планує відкрити код JavaFX і продовжити його розвиток в рамках підпроекту [OpenJDK](https://uk.wikipedia.org/wiki/OpenJDK). Код JavaFX, як і код OpenJDK, розповсюджуватиметься під ліцензією [GPL](https://uk.wikipedia.org/wiki/GPL) v2 з винятками [GNU ClassPath](https://uk.wikipedia.org/w/index.php?title=GNU_ClassPath&action=edit&redlink=1), що дозволяє динамічне зв'язування з комерційними продуктами. Oracle JavaFX runtime і SDK і раніше будуть доступні тільки в бінарному вигляді, під ліцензією BCL (Binary Code Licence).

Як середовище розробки було обрано IntelIJ IDEA 2018.3.

IntelliJ IDEA — комерційне [інтегроване середовище розробки](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D1%81%D0%B5%D1%80%D0%B5%D0%B4%D0%BE%D0%B2%D0%B8%D1%89%D0%B5_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B8) для різних мов програмування ([Java](https://uk.wikipedia.org/wiki/Java), [Python](https://uk.wikipedia.org/wiki/Python), [Scala](https://uk.wikipedia.org/wiki/Scala), [PHP](https://uk.wikipedia.org/wiki/PHP) та ін.[[3]](https://uk.wikipedia.org/wiki/IntelliJ_IDEA#cite_note-:0-3)) від компанії [JetBrains](https://uk.wikipedia.org/wiki/JetBrains). Система поставляється у вигляді урізаної по функціональності [безкоштовної](https://uk.wikipedia.org/wiki/%D0%91%D0%B5%D0%B7%D0%BF%D0%BB%D0%B0%D1%82%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F)версії «Community Edition» і повнофункціональної [комерційної](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B5%D1%80%D1%86%D1%96%D0%B9%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F) версії «Ultimate Edition», для якої активні розробники відкритих проектів мають можливість отримати безкоштовну ліцензію. [Сирцеві тексти](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%80%D1%86%D0%B5%D0%B2%D1%96_%D1%82%D0%B5%D0%BA%D1%81%D1%82%D0%B8) Community версії поширюються рамках [ліцензії Apache 2.0](https://uk.wikipedia.org/wiki/%D0%9B%D1%96%D1%86%D0%B5%D0%BD%D0%B7%D1%96%D1%8F_Apache). Бінарні збірки підготовлені для [Linux](https://uk.wikipedia.org/wiki/Linux), [Mac OS X](https://uk.wikipedia.org/wiki/Mac_OS_X) і [Windows](https://uk.wikipedia.org/wiki/Windows).

Також потрібно розглянути основні поняття для вибраної теми.

Скінченний автомат (або просто FSM – Finite-state machine) це модель обчислень, заснована на гіпотетичної машині станів. В один момент часу тільки один стан може бути активним. Отже, для виконання будь-яких дій машина повинна змінювати свій стан.

# 4 АЛГОРИТМ РІШЕННЯ ЗАДАЧІ

На основі заданої регулярної граматики можна побудувати еквівалентний їй скінченний автомат і, навпаки, для заданого скінченного автомату можна побудувати еквівалентну йому регулярну граматику.

Регулярні граматики використовуються для визначення лексичних конструкцій мов програмування. Створивши автомат на основі відомої граматики, ми отримаємо розпізнавач для лексичної конструкції даної мови. Таким чином вдається вирішити задачу розбору для лексичних конструкцій мови, заданих довільною регулярної граматики.

Задача формується наступним чином: є ліволінійна граматика, що задає мову, необхідно побудувати еквівалентний їй скінченний автомат, що задає ту ж мову. Задача вирішується в два етапи:

* вхідну ліволінійну граматику потрібно перевести до автоматного вигляду;
* на основі отриманої автоматної ліволінійної граматики будується шуканий автомат.

Алгоритм перетворення до автоматного виду:

* побудувати множину нетермінальних символів;
* почати продивлятись множину правил Р граматики G, для правил що мають більше ніж один термінальний символ створюються нетермінальні символи та правила з цими символами, які містять по одному термінальному символу, правила що містять тільки термінальний символ, або по одному термінальному та нетермінальному символу переносяться до множини правил Р автоматної граматики;
* продивлятися множину правил Р граматики G. Якщо знаходиться правило виду А→В, якщо в ньому присутні правила виду В→С, В→ Са, В→ а то в нього додаються правила виду А→С, А→ Са, А→ а (при цьому потрібно враховувати, що в граматиці не повинно мати однакових правил, і якщо якісь правила вже присутні в граматиці, то повторно його додавати не потрібно);
* якщо на кроці 3 не було знайдено хоча б одне правило вигляду А→В або де є порожня множина то потрібно повторити крок 3, або ж перейти до кроку 5;
* цільовий символ граматики G стає символ S.

Алгоритм побудови скінченного автомату на основі ліволінійної граматики:

* будуємо стани автомату Q. Стани автомату будуються таким чином, щоб кожному нетермінальному символу з множини нетермінальних символів граматики співпадав один стан з множини Q автомату. Крім того, в множину станів автомату додається ще один додатковий стан, що буде позначатися H. Зберігаючи позначення нетермінальних символів граматики G, для множини станів автомату;
* вхідним алфавітом автомату М є множина термінальних символів граматики G;
* продивляємось всі множини правил вхідної граматики. Якщо зустрічається правило виду А→ а, то в функцію переходів автомату додаємо стан (Н, а) = А. Якщо зустрічається правило виду А→ Ва, то в функцію переходів автомату М додаємо стан (В, а) = А;
* початковим станом автомату М є стан Н;
* множина скінченних станів автомату М складається з одного стану. Цей стан є станом, співпадаючи цільовому символу граматику.

Алгоритм роботи програми зображено на рисунку 4.1:

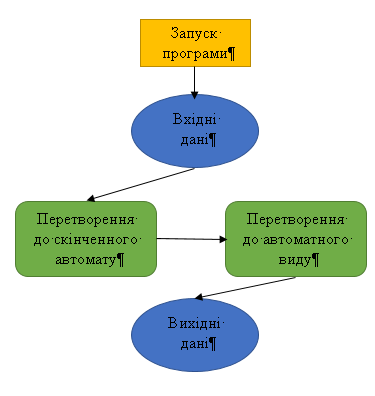


Рисунок 4.1 – Алгоритм роботи програми

Програма поділяється на основні класи для реалізації даних алгоритмів.

Клас AutomationGramar слугує для створення об’єкту автоматної граматики, в ньому створені змінні, а також методи для задання та отримання цих змінних.

Клас RegGramar слугує для створення об’єкту регулярної граматики, в ньому створені змінні, а також методи для задання та отримання цих змінних.

Клас RegularInAutomation слугує для розміщення методів перетворення регулярної граматики до автоматного виду, а також об’єктів класу регулярної граматика та автоматної граматики.

Клас StateMachine слугує для створення об’єкту скінченного автомату який містить змінні автомату, а також методів для заповнення змінних та отримання змінних.

Клас GenerateSM слугує для розміщення методів перетворення з автоматної граматики до скінченного автомату, а також об’єктів класу автоматної граматики та скінченного автомату.

Клас TransitionFunction слугує для розміщення кроків перетворення спочатку з регулярної граматики до автоматної, а пізніше з граматики автоматного виду до скінченного автомату.

Також в програмі створені контролери, для зв’язку візуального представлення з програмними діями, і файли XML в якому описуються візуальні представлення програми.

# 5 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

Для запуску даного програмного забезпечення потрібно щоб на комп’ютері було встановлено JDK, після чого запускаємо jar файл, після чого перед нами з’явиться головне меню програми, зображено на рисунку 5.1.

Програма дозволяє ввести власні нетермінальні символи, термінальні символи, цільовий символ, а також заповнити граматики власними правилами з термінальними та нетермінальними символами, що задав користувач. Якщо натиснути на кнопку «Додати правило» до граматики буде додано правило, якщо воно співпадає з правилами запису регулярних граматик. Натиснувши на кнопку «Приклад», граматика автоматично заповниться нетермінальними символам, термінальними символами, цільовим символом а також правилами регулярної граматики механічного годинника, користувач може додавати свої символи та правила. Натиснувши кнопку «Перетворити до СА», ми потрапимо до меню скінченного автомату, програма автоматично перетворить регулярну граматику до автоматного виду, після чого перетворить її до скінченного автомату.



Рисунок 5.1 – Головне меню програми

В меню скінченного автомату, зображено на рисунку 5.2, користувач має можливість продивитися автоматну граматику, яка була отримана шляхом перетворення регулярної граматики до автоматного виду, а також запис скінченного автомату, також для користувача буде наведено список правил автоматної граматики, а також функції переходів для скінченного автомату. Також користувачу буде доступно три кнопки. Натиснувши на кнопку «Показати граф СА» користувач зможе продивитися граф скінченного автомату. Натиснувши кнопку «Кроки переведення до СА» користувач може продивитися кроки переведення до скінченного автомату. Натиснувши кнопку «Показати приклад» ми продивимось графічний приклад механічного годинника.

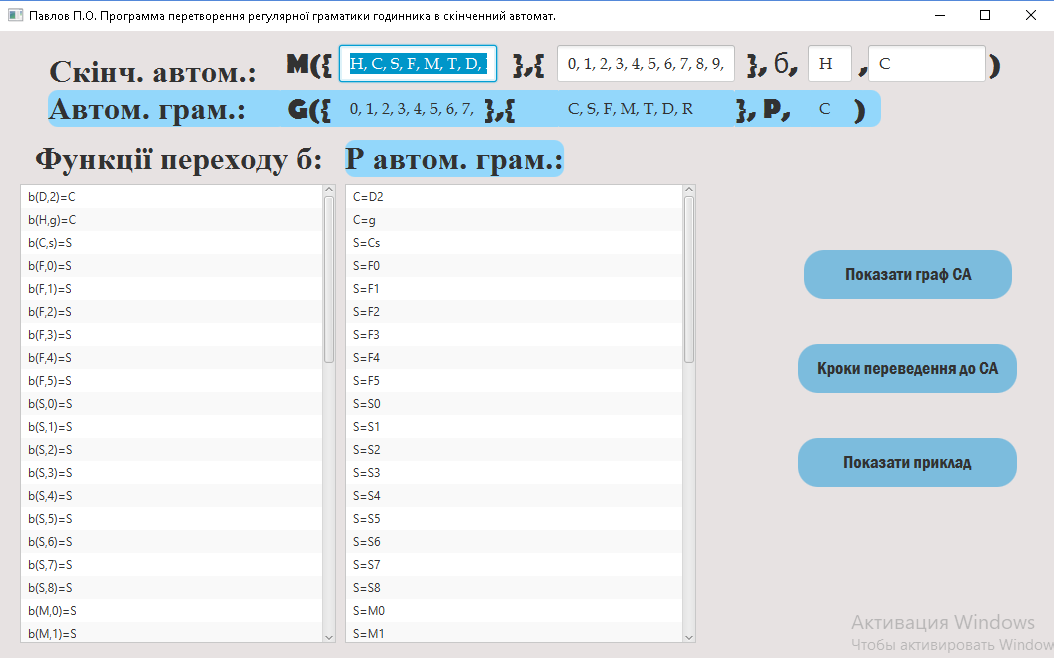


Рисунок 5.2 – Меню скінченного автомату

На основі функцій переходів ми можемо побудувати граф скінченного автомату. В даній програмі граф малюється за допомогою HTML сторінки а також мови програмування JavaScript, після чого відображається як веб-сторінка в окремому вікні програми. Граф — це сукупність об'єктів із зв'язками між ними. Об'єкти розглядаються як вершини, або вузли графу, а зв'язки — як дуги, або ребра. Для різних областей використання види графів можуть відрізнятися орієнтованістю, обмеженнями на кількість зв'язків і додатковими даними про вершини або ребра. Граф скінченного автомату, зображено на рисунку 5.3.

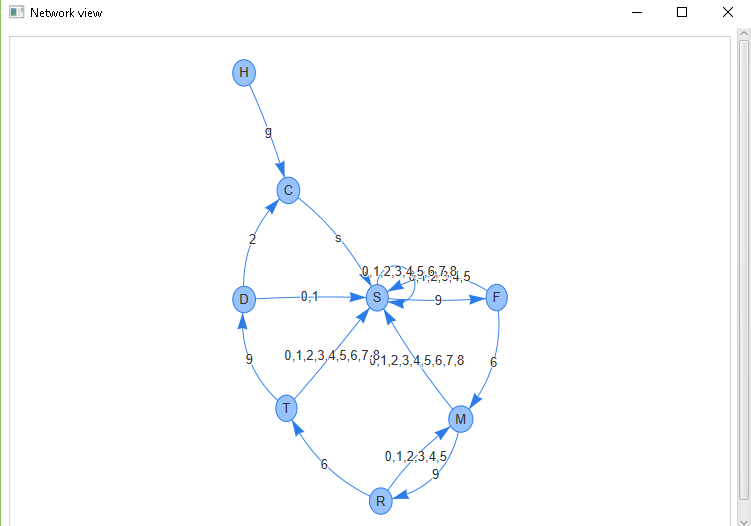


Рисунок 5.3 – Граф скінченного автомату

У вікні кроків переведення до скінченного автомату, записані дії які виконувала програма спочатку для переведення до автоматного виду, програма записувала всі зміни, а також кроки створення автоматної граматики, потім дії переведення з автоматної граматики до скінченного автомату, додавання даних, а також створення функцій переходів. Вікно кроків зображене на рисунку 5.4.

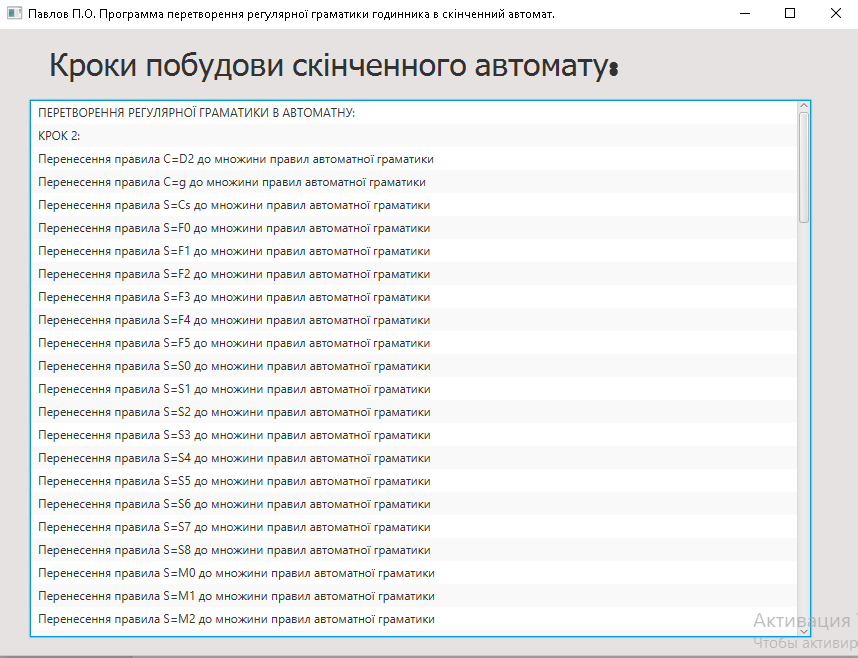


Рисунок 5.4 – Кроки побудови скінченного автомату

Для показу роботи скінченного автомату в програмі було створено приклад його використання. Найбільш застосовуваним в побуті пристроєм є механічний годинник, тому саме йог обуло створено з використанням скінченного автомату. Годинник був створений за допомогою HTML та стилізований за допомогою каскадних таблиць CSS, для створення дії годинника за допомогою скінченного автомату було створено анімації. Зображений приклад на рисунку 5.5.



Рисунок 5.5 – Приклад реалізації скінченного автомату

# ВИСНОВКИ

На переддипломній практиці було розглянуто основні поняття теорії скінченних автоматів. Була розроблена програма яка перетворює задану користувачем регулярну граматику до скінченного автомату, показує граф скінченного автомату та шляхи отримання результату. Як приклад було взято скінченний автомат механічного годинника.

Отже, скінченний автомат – в теорії алгоритмів математична абстракція, що дозволяє описувати шляхи зміни стану об'єкту залежно від його поточного стану і вхідних даних, за умови, що загальна можлива кількість станів скінченна.

Скінченні автомати широко використовуються на практиці, наприклад в синтаксичних, лексичних аналізаторах, і тестуванні програмного забезпечення на основі моделей.

Мислення в термінах скінченних автоматів (тобто розбиття виконання програми на кроки автомата й передача інформації від кроку до кроку через стани) необхідно при побудові об'єктно-орієнтованих додатків. У цьому випадку програмування в стилі скінченних автоматів виявляється єдиною альтернативою породженню множин процесів або потоків управління.

Часто поняття станів і машин станів використається для специфікації програм. Так, при проектуванні програмного забезпечення за допомогою UML для опису поведінки об'єктів використаються діаграми станів. Крім того, явне виділення станів використовується в описі мережних протоколів.

Виконавши дану роботу можна зробити висновок, що застосування скінченних автоматів, для побудови програмних рішень, є дуже актуальне та надійне в сучасному світі інформаційних технологій.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Альфред, В. Ахо Компиляторы. Принципы, технологии и инструментарий / Альфред В. Ахо и др. - М.: Вильямс, 2015. - **266** c.
2. Дехтярь М.И. Введение в схемы, автоматы и алгоритмы. - М.: Наука, 2002. С. 642.
3. Коган Д.И., Бабкина Т.С. Концепции конечного автомата и регулярного языка. Операции над регулярными языками. М.: Наука, 2000.
4. Шилдт,Герберт. Java 8: полное руководство.: Пер. с англ. — М.: ООО *"*И.Д. Вильямс*"*, 2011. — 1056 с.: ил. — Парал. тит. англ.
5. Хорстман, Кей С., Корнелл,Гари. Java Библиотека профессионала, том 1. Основы 9-е изд. : Пер. с англ. – М. : ООО «И.Д.Вильямс»,2014. – 864с.
6. Хорстман, Кей С., Корнелл,Гари. Java Библиотека профессионала, том 2. Расширенные средства 9-е изд. : Пер. с англ. – М. : ООО «И.Д.Вильямс»,2014. – 1008с.
7. Professor Web [Електронний ресурс] – Режим доступу до ресурсу: https://professorweb.ru/.
8. Metanit.com [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/cpp/tutorial/
9. Cyberforum [Електронний ресурс] – Режим доступу до ресурсу: http://www.cyberforum.ru/
10. Devcolibri [Електронний ресурс] – Режим доступу до ресурсу: https://devcolibri.com
11. StackOverflow [Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com/