Недетерминиран Краен Автомат, JAVA

Момчил Милков

1. Описание и цели на проекта

Да се реализира програма, която поддържа операции с недетерминиран краен автомат с Ɛ-преходи, над азбука, състояща се от цифрите и малките латински букви. Автоматите да се сериализират. Всеки прочетен автомат да получава уникален идентификатор. Да се реализира мутатор, който детерминира даден автомат, както и операция, която проверява дали езикът на даден автомат е краен.

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции:

open	Отваря съществуващ автомат за работа
close	Затваря отворен автомат
save	Записва нов автомат
save as	Записва автомат в определен път
help	Извеждане на всички команди в програмата
exit	Излизане от програмата
list	Списък с идентификаторите на всички автомати
print	Извежда информация за всички преходи в автомата
empty	Проверява дали езикът на автомата е празен
deterministic	Проверява дали автомат е детерминиран
recognise	Проверява дали дадена дума е в езика на автомата
union	Намира обединението на два автомата и създава нов автомат
concat	Намира конкатенацията на два автомата и създава нов автомат
un	Намира позитивна обвивка на автомат и създава нов автомат
reg	Създава нов автомат по даден регулярен израз

-функционалностите в червен текст не са реализирани в проекта*

2. Основни дефиниции

Крайните автомати четат редица от символи (от входна азбука), наречена входна дума, и извеждат също редица от символи (от изходна азбука), наречена изходна дума. Множеството от всички входни думи се нарича език, разпознаван от автомата.

В зависимост от програмата си, крайният автомат притежава определен краен брой състояния, в които може да се намира. Начално състояние е състоянието, в което се намира автомата при започване на програмата.

Работата на крайните автомати се състои от определен брой стъпки, като на всяка стъпка се чете следващият символ на входната дума. В зависимост от прочетения символ и състоянието, в което се намира, автоматът извежда редица от изходни символи и преминава в ново състояние.

Математически, крайните автомати са са представени като M = (S, Σ, T, I, A), където:

• S е множеството на състоянията на автомата

• Σ е азбуката на автомата

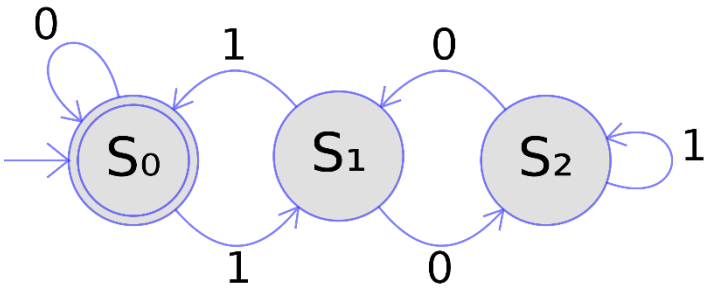
• T е множеството на преходите между състоянията на автомата: (p, x, q) ∈ T където (p, q) ∈ SxS и x ∈ Σ

• I е множеството на началните състояния [1] I⊆S

• A е множеството на крайните състояния на автомата. Това са състояния, които позволяват „излизане“ от автомата. A⊆S

Крайните автомати биват детерминирани и недетерминирани. При детерминираните може да има най-много по един преход от дадено състояние за всяка буква от азбуката на автомата, докато при недетерминираните са възможни няколко различни прехода за една и съща буква. Също така детерминираните крайни автомати имат едно-единствено начално състояние. Всеки недетерминиран автомат може да се преобразува в детерминиран.

Крайният автомат може да се представи като ориентиран граф, или чрез таблица:



	a	b
{1}	{1,2}	{1}
{1,2}	{1,2,3}	{1,3}
{1,2,3}	{1,2,3}	{1,3}
{1,3}	{1,2}	{1}

3. Реализация

Почти всяка функционалност в проекта се реализира от специален конкретен клас, притежаващ статичен публичен метод, който обработва подаден автомат. За визуализация и навигация на потребителя се използват три меню класа: HelpMenu – чрез извикване на неговия метод help(), се извеждат всички команди в програмата; MainMenu – главното меню в програмата, където се намираме в начално положение и имаме редица опции (help, open, list, build, union, concat, exit), които не касаят конкретни инстанции на автомати (реализирано чрез switch); AutomataMenu – идентично на MainMenu, с разликата, че за да влезем в това меню трябва да използваме команда open (от MainMenu), чрез която отваряме конкретен автомат. От там имаме редица нови опции касаещи автомата (print, saveas, recognise, close).

3.1 Automata

Основният клас в програмата е класът „Automata" (автомат). Върху негови инстанции ще бъдат извършвани повечето операции от проекта. Както в горе- описаният математически модел, нашият автомат съдържа полета за: начално състояние(String firstState), множество от състояния(List<String> states), множество от променливи/азбука(List<String> variables), множество от крайни състояния(List<String> endStates), и таблица на преходите(String[][] table). Освен това, класът притежава и конструктор, в който се извикват две външни функции от два други класа, които служат за изграждане на нов автомат.

3.2 AutomataMaker

В този клас, потребителят въвежда всичката информация нужна за изграждането на автомата, като се следи за най- различни грешки във въвеждането (повторение на състояния, създаване на преходи към несъществуващи състояния, празни символи и др.). След като цялата информация е записана, тя се присвоява на полетата на новата инстанция на автомат(от 3.1).

3.3 AutomataDeterminer

След като имаме готовият автомат, следва да проверим дали е детерминиран. Това е нужно, защото всички операции над автомати в програмата са проектирани да функционират само върху детерминирани автомати. Този клас прави проверката, като ако автоматът вече е детерминиран, конструкцията завършва, а ако не е, бива детерминиран чрез следният алгоритъм:

1. В таблица на преходите записваме преходите на началното състояние на автомата.
2. За всеки преход от новият ред на таблицата, ако комбинацията от различните опции все още не е срещната в таблицата, я считаме за ново състояние и я добавяме като нов ред. Записваме преходите на новото състояние което сме добавили.
3. Повтаряме стъпка 2, докато не са останали никакви непознати състояния и получаваме детерминираната таблица на старият недетерминиран автомат.

Според новата таблица се обновяват всички полета на въпросния автомат.

	a	b
{1}	{1,2}	{1}
{1,2}	{1,2,3}	{1,3}
{1,2,3}	{1,2,3}	{1,3}
{1,3}	{1,2}	{1}

3.4 Сериализация/Десериализация (AutomataWriter/AutomataReader)

След създаване на автомат, в клас AutomataWriter, програмата изисква идентификатор под който автомата да бъде записан. Идентификаторите са уникални за всеки автомат. Обектът се записва във файл, чрез ObjectOutputStream, а за име на файла се дава идентификатора.

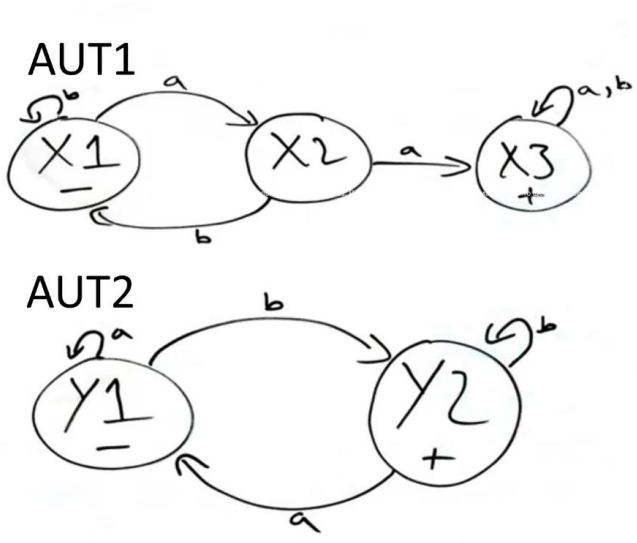
Аналогично, чрез AutomataReader, програмата чете подаден идентификатор, като ако съществува файл с такова име, чрез ObjectInputStream той се десериализира. Допълнително, всеки прочетен в текущата сесия автомат, се записва в HashMap, като по този начин, при следващи извиквания на същия автомат, няма нужда отново да го търсим и четем от файл, а директно от HashMap-а.

3.5 WordRecogniser

Чрез този клас, автоматите разпознават подадени към тях думи. Подадената дума се разбива на отделни символи, като чрез всеки последователен символ се проследява пътя по таблицата на автомата (дума: aba, от S1 по a: S2, от S2 по b: S3, от S3 по a: S2). Ако, последното състояние от поредицата от преходи се окаже крайно състояние за автомата, думата е разпозната. Иначе, думата е отхвърлена.

3.6 AutomataUnion

При търсене на обединението на два автомата, за начално състояние се взима конкатенацията на началните състояния на двата автомата. Според новото начално състояние, запълваме таблицата на преходите, подобно на AutomataDeterminer(3.3), но тук, към състоянието след прехода конкатенираме резултатите от всички състояния преди прехода(от x1y1 по a, резултатът е x2y1, защото x1 по a влиза в x2, y1 по a влиза в y1). Всяка нова комбинация считаме за ново състояние в автомата и повтаряме операцията. Освен това на всяка комбинация присвояваме нов символ за да се улесни четимостта (x1y1 = z1, x2y1 = z2). За крайни състояния се считат всички състояния, в които присъства крайно състояние от който и да е от двата начални автомата.



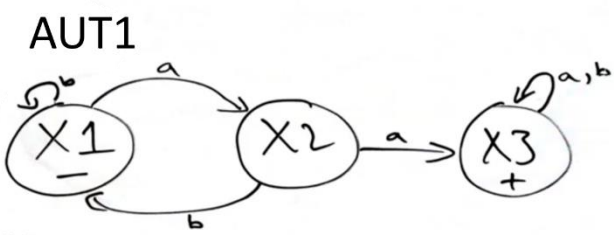
OLD STATE	a	b
$X_1Y_1 = Z_1$	$X_2Y_1 = Z_2$	$X_1Y_2 = Z_3$
$X_2Y_1 = Z_2$	$X_3Y_1 = Z_4$	$X_1Y_2 = Z_3$
$X_1Y_2 = Z_3$	$X_2Y_1 = Z_2$	$X_1Y_2 = Z_3$
$X_3Y_1 = Z_4$	$X_3Y_1 = Z_4$	$X_3Y_2 = Z_5$
$X_3Y_2 = Z_5$	$X_3Y_1 = Z_4$	$X_3Y_2 = Z_5$

За целта се използват общо три класа. Първоначално се подават двата автомата, на които ще търсим обединението. Тяхната информация се обработва от клас Initializer, който извършва редица операции, нужни за реализацията на обединението в java. След това, приготвените обекти и структури влизат в клас AutomataUnion (главният клас на операцията), където се изпълнява по- горе описания алгоритъм. След като обединението е намерено, изходната информация влиза в клас Product, който я преобразува в подходящ формат за да може тя да бъде записана под формата на нов Автомат. На края на операцията, се връща новият автомат, изграден от обединението на двата оригинални. Initializer-**AutomataUnion**-Product.

-Забележка: в програмата не може да се търси обединението на автомат с празни преходи*

3.7 AutomataConcatenation

При търсене на конкатенацията на два автомата, за начално състояние взимаме началното състояние на първия подаден автомат. Започваме да запълваме таблицата на преходите както в 3.3 и 3.6. Ако стигнем до крайно състояние на AUT1, в таблицата го записваме конкатенирано с началното състояние на AUT2 (x3 = x3y1). Аналогично на 3.6 запълваме останалата част от таблицата, и на всяко уникално състояние присвояваме нов символ. За крайни състояния на новия автомат взимаме всички състояния в които присъства крайното състояние на втория автомат.



OLD STATE	a	b
$X_1 = Z_1$	$X_2 = Z_2$	$X_1 = Z_1$
$X_2 = Z_2$	$X_3 Y_1 = Z_3$	$X_1 = Z_1$
$X_3 Y_1 = Z_3$	$X_3 Y_1 = Z_3$	$X_3 Y_1 Y_2 = Z_4$
$X_3 Y_1 Y_2 = Z_4$	$X_3 Y_1 = Z_3$	$X_3 Y_1 Y_2 = Z_4$

За реализацията в java, използваме същата 3-класова структура както в 3.6, като тук същинската операция, описана по- горе се изпълнява от клас AutomataConcatenation: Initializer-**AutomataConcatenation**-Product.

-Забележка: в програмата не може да се търси конкатенацията на автомат с празни преходи*

3.8 Допълнителни класове и функции

3.8.1 AutomataLister – Извежда списък от идентификаторите на всички автомати (от файловете)

3.8.2 AutomataPrinter – Извежда на екрана таблицата на преходите на текущия автомат, заедно с крайните му състояния

3.8.3 SaveAs – Записва текущия автомат в директория и с име, избрани от потребителя

3.8.4 EmptyLang – Проверява дали езикът на текущия автомат е празен

3.8.5 CopyAutomata – Копиране на съществуващ автомат върху нов

4. Забележки и заключение

Поради редица технически затруднения, не всички функционалности от условието са реализирани, като около 90% от оригиналното задание е изпълнено. Имайки предвид сравнително абстрактната материя поставена в него, повечето методи са разработени напълно оригинално (липса на точки на сравнение в интернет и литературата), тоест е високо- вероятно да съществуват по- оптимални решения. Понеже програмата се фокусира върху един единствен обект – Автомата и операции върху него, за цялостната архитектура е избран модел, при който всяка една функционалност се изпълнява от отделен клас, съдържащ статичен метод. В крайна сметка, програмата използва 18 класа, като този модел осигурява високо разделение на отговорностите, както и лесно добавяне и премахване на функционалности.

5. Използвана литература и референции

- Курс по Логика и Автомати, ТУ Варна
- Курс по Дискретни Структури, ТУ Варна
- Курс по Организация на Компютъра, ТУ Варна
- Обединение на Автомати(теорема на Клини): https://www.youtube.com/watch?v=_SYpSmOxEWQ
- Конкатенация на Автомати(теорема на Клини): https://www.youtube.com/watch?v=i5-TQnLDKQ0