Maruf Abdirimov

  Automata and Logic Engineering 2

Work Report

Table of contents

**Элементы оглавления не найдены.**

# Introduction

This is a work report for a course of Automata and Logic Engineering 2(ALE2). In this report, following topics will be discussed:

* Assignments and how they were solved
* Software Design
* System Interface
* Testing
* Conclusions and recommendations

# Parse & DFA

For this assignment 3 different classes were created, each serving as a standalone static service-provider, meaning all methods in the class were static and could be used without the object.

First class was **Parser**. This service’s responsibility was reading a file from the provided path and transforming it into an Automaton object. To make class easy to read and maintain, main method was further divided into smaller methods responsible for parsing specific lines in the documents with reusability in mind. So, for example, method used to parse line of alphabet can also be used to parse line with stack.

Second class, **IdentifierDFA,** was responsible for checking whether Automaton is DFA or not.

And final third class, had 2 functions. First turning Automaton into a txt document ready to be turned into a graph, and second actually turning that txt file into picture, using GraphViz.

# Read Word / Accept String

The second assignment was implemented by creating two static methods in a new class/service:

* **checkWord**
* **doesNotGoToFinal**

**checkWord** works recursively by going through the states search for all the possible transitions from this state and tries to find a path so that the whole word can be accepted and if the final state is not reached but the word is read then another method is called **doesNotGoToFinal** which check if from a current state it is possible to go to the final state using empty transitions, if from the given state the automata goes to a final state the word is accepted.

The implementation is not the most efficient one because it searches for a word by using brute force to try every possible path. The only optimization made is that if the word is empty it directly goes to the method **doesNotGoToFinal** and only checks if from the first state you can go to the last with empty transitions.

# Regular Expression

For the third assignment a new service was made with static methods. To read regular expression decision was made to first create methods that read individual letters and then reuse it methods for reading logical expressions(\*|.). After which create a method that will be responsible for deciding which method to call depending on the string passed to it.

And all that left afterward, was a process of testing and tweaking based on the tests’ results.

# Finite Check

The assignment was straight forward to be solved if there is a loop then it is not finite.

There are two kinds of loops, self-loop or a loop which consists from several states. To detect the self-loop was really easy just check if there is a transition which goes to itself. But it was more challenging to find a loop between states for that purpose a list with passed transitions was created. When going through all the possible transitions recursively the method was checking if a transition repeats then it is part of a big loop and the automata is not finite.

But checking for loops was only one part of the solution to make sure that this loops is actually part of a path which goes to final a method which was created earlier was reused **doesNotGoToFinal** by passing the state where the loop was final and checking if it goes to final state this proves that the automata is finite or not.

The last part of the assignment was to generate all the words from a finite automaton and to make this functionality possible anther methods was created which generates words from an automata. It goes through the states recursively and try to find all the possible words without repeating them after they are generated, they are displayed in the GUI.

# NDFA to DFA

To implement solution for this assignment, following video was used <https://www.youtube.com/watch?v=taClnxU-nao&list=PLWVn8G3ruRW7bDI16wpW-dO0FMwR4yjCu&index=25&t=585s>

New class/service was made to accomplish required functionality.

First initial table was created filling up all of the transitions object has. Next, second fills up based on the values provided in the first one. And finally, all of that data is transformed into a new Deterministic Automaton.

However, methods work perfectly only with states that follow specific naming. So that all the names of the states should be ascending positive integers.

# PDA

To implement this assignment, it was quite easy because not a lot of code had to be changed. The parser was changed, and in the check words several things were added.

The parser was changed to check if there is a stack word in the input file then the transitions were parsed in a different way so the PutOnStack and TakeFromStack are initialized according to the file input. The previous word check was reused to work with the stack.

One additional method was added in automata which is **doesNotGoToFinalPDA** its works on the same principal as before but takes the stack into consideration. I could not reuse the previous one because a lot of things had to be changed because it was used to check for if the automata is finite and it was breaking a lot of things.

In the future more time can be spent to explore a solution where the method is reused and not repeated with some modifications to make less repeating code.

# Software Design

## System Overview

Priority was to create standalone service capable of doing all of the functionality in their narrow specialty. Thus all services have separate classes with static methods. Moreover to accommodate for the fact that these are services, all the classes were made in the library which was then referenced in the Forms and Testing Projects.

All 3 basic classes have their Equals and ToString methods overridden to increase flexibility of the classes and allow dynamic comparison between them, as well a quick way to transform object into a string-type description of the object.

## System Architecture

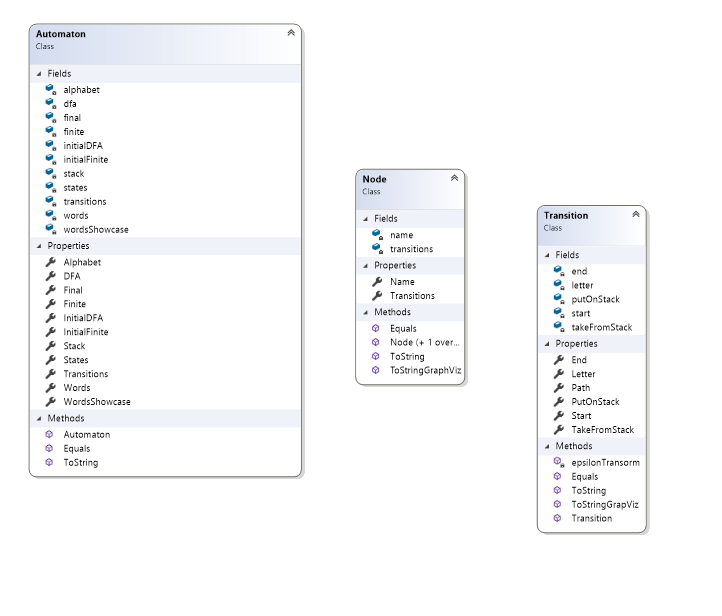


Figure 1. Base Classes

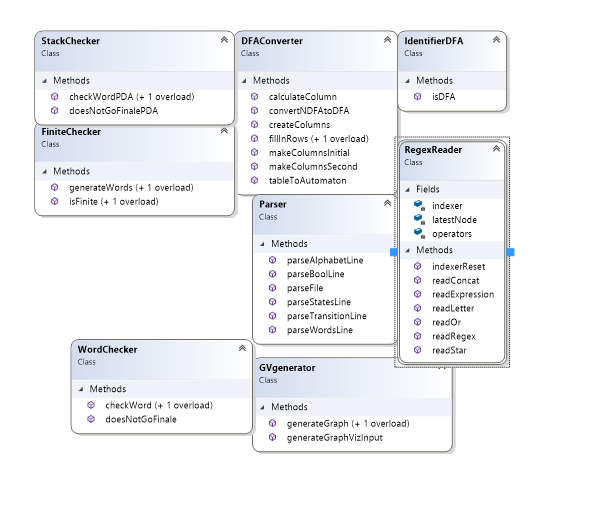


Figure 2. Service Classes

# Interface

Due to time constraints, interface was made to work and not to satisfy aesthetical needs of a person. Which means, that idea behind it was to ensure all of the main functionalities bound to respective buttons and events.

# Testing

Because Project was in done in a Test Driven Development(TDD) style, tests were prioritized and were written before the actual implementation. Exceptions were Assignment 2,4,6 because me and Veselin Nasev were pair-programming on his machine and he does not like TDD.

Unfortunately, due to time constraints only 63 tests have been done on the moment of writing this report. Especially undertested were services done for Assignment 2,4 and 6, due to the reasons mentioned above.

The overall code coverage for the services in the library is 91% with around 7% being purely comfort-related functions used for overloading and having no real functional importance. As for models, they were covered only by 62%, mainly due to how many variables there were in the Automaton, that didn’t require testing.

All of tests follow standard naming conventions:

{MethodName}Test\_{ArgumentsPassedToMethodWithDetails}\_{ExpectedBehaviour}

Example: **checkWordPDATest\_passWrongAutomaton\_throwsException**

Unfortunately, due to low experience in TDD(only half-a-year), I learned about incredible testing tools for Visual Studio(ReSharper) only two days before the deadline. Otherwise, progress of the project would have so much more efficient.

# Conclusions and recommendations

This project was challenging. Something that made think and made you appreciate your own effort in case you are successful. Moreover, chance to work with friend and think over a problem, trying to come up with a solution was a very enjoyable experience.

However, that’s where good points end. This subject is worth 3ECs or maximum 28 hours \* 3 on the whole subject. So, on average this subject should take 80 hours to finish, same as subjects like PRC1 or IPV. But I spend more hours on this subject than on PRC1 and IPV combined, while working in pair with my friend.

I feel like this subject is worth less ECs than it should. Based on the hours alone I feel like I should definitely pass this subject with flying colors. But the problem is that I am not even close to being satisfied with the work I will be delivering. There are so many things I’d want to improve, functionalities to test and ideas to implement, that I will at the very least require 2 if not all 40 more hours to make it a wholesome product.

But that aside, lessons were interesting and teacher, put in real effort to help us understand what is required from us to finish the course. In Fontys, that is rare and appreciate this. I hope, teacher remains as spirited as she was in this semester and keep bringing light of ALE2 to the next year’s students.