Rheinisch-Westfälische Technische Hochschule Aachen
Chair of Process and Data Science
Prof. Dr. Wil van der Aalst

Process Conformance Checking in Python in SS 2021

Alignments on NFA(s) in Micropython

Group:3

Sprint 1

04.06.2021

supervisor: Alessandro Berti

Contents

List of Figures

1 Introduction

The goal of this project is to implement conformance checking in python with no dependencies. In order to achieve it, in this first sprint we have started with converting regular expressions into NFAs, implementation of NFA model and checking their fitness. Our focus was to achieve these tasks keeping in view the optimality conditions for memory usage and runtime. This document covers a brief summary of what we had planned and how we have achieved it in the current sprint. Chapter 2 discusses various aspects like work distribution, implementation of the planned tasks, testing and documentation. Chapter 3 is comprised of phase reviews of the entire team. As far as testing is concerned, we have begun with manual testing in this sprint which will be replaced by unit testing in the upcoming sprints.

2 Summary of Sprint 1

2.1 Work Distribution

The team met at the start of the sprint and discussed about the deliverable of Sprint 1. There were no hard assignments of roles in this sprint and the every member contributed in the development of the code base. The tasks of this sprint were divided in to following major phases,

- Theory Analysis
- Conversion of Regular Expression to NFA
- Implementation of NFA Model
- Fitness Checking
- Quality Assurance
- Bug Fixing
- Documentation

The Trello board shows the development cycle for this sprint.

2.2 Implementation

As this sprint was the first one in which we started coding, we decided to lay a good foundation for the rest of the project and take our time to test different implementation ideas. We decided on focusing on creating the data model for an nfa as well as the transformation from a regular expression into an nfa. The reason we did so is that we determined these as the base concepts that build the foundation for the conformance checking we are planning to do.

2.2.1 Nfa data model

Nfa are non-deterministic finite automaton that were first introduced by Michael O. Rabin and Dana Scott in 1959 [2]. Formally they are described by a tuple of 5 parts (Q, Σ, Δ, q0, F).

With Q being the finite set of states.
Σ being the finite set of input symbols.
$\Delta : Q\Sigma \rightarrow P(Q)$ being the transition function where it is possible to have non deterministic behaviour such that an input can have multiple places as the target from a given start place.
q0 being the initial start place. F is a set of places that are allowed final states.

First, we created a data model that mimicked this tuple of five elements. Obviously there are multiple duplications and inefficiencies when storing the nfa like that, so we came up with multiple iterations of improvements. As defined in the requirements we had mainly two metrics we wanted to optimize: Memory usage and run time.

The main way we reduced the memory usage was by removing duplicated information. As a result the nfa data model does not store the set of events separately but instead it is indirectly given as the union of the events that occur in all the transitions.

The main way we increased the performance of for example the function that checks whether a trace can be replayed by the given nfa model is that we does not store all possible transitions together at the nfa level, but instead each place in the nfa stores all the transitions that are connected to it. As each transition has only one start node this creates no duplication and is therefore equally memory efficient. This way of storing the transitions improves the performance because when replaying a trace on a model one is only interested in possible transitions connected to the current place and not in transitions not connected to the current place.

Here we wanted to point out that this also creates the possibility to reduce the memory usage of an nfa further by not having to store the start place with the transition as this is always defined to be the place that has a reference to the transition. For this sprint we decided not to include this last memory saving approach because we wanted to first evaluate that this does not decrease the performance of the conformance checking algorithm we are going to implement in the following sprint.

In addition we would like to point out that we added the possibility to use epsilon transition in the nfa model which is then called an ϵ - nfa. For simplicity reasons we are going to write nfa instead of ϵ - nfa as there exist a conversion between them any ways and it just shorter to read and write.

2.2.2 Transformation from a regular expression into a nfa

The proof that regular expressions and nfas (ϵ - nfas) can represent the same languages is constructive one that is known as the Thompson's construction [3]. This describes rules to convert any regular expression into an ϵ - nfa.

But before the algorithm can apply the construction rules, the regex need to be parsed so that the program understands the regular expression in a correct manner. To do so, we evaluated to different approaches. One being a conversion from the infix notation of a regular expression into the prefix notation [5]. The second being a parsing approach based on a recursive descent parser [4].

In the end we decided to go with the recursive descent parser due to the smaller memory usage. The infix to prefix notation conversion approach used a lot of memory as it was required to keep a stack with all the previous nfa models as well as the previous sub parts of the regular expression string. In addition, another benefit of the recursive descent parser is that is very closely related to a grammar that describes a regular expression. Therefore

we was able to very concretely define the form of a regular expression that we are accepting:

Expression := Konkat { "|" $Konkat$}
$Konkat := Prod\{(".","")Prod\}$
$Prod := Factor(" * ","")$
$Factor := Activity$ | "(" $Expression$")"
$Activity :=$ "a" | "b" | ...($allletters$)

This grammar based approach is also future proof as we extend the grammar of the used regular expressions it is straight forward to extend the parsing functionality as well.

Once the input is parsed and interpreted correctly the algorithm can follow the rules defined by Thompson's construction to construct the resulting nfa by merging smaller base cases together:

The base case creates an nfa for a single event a that could also be an epsilon:

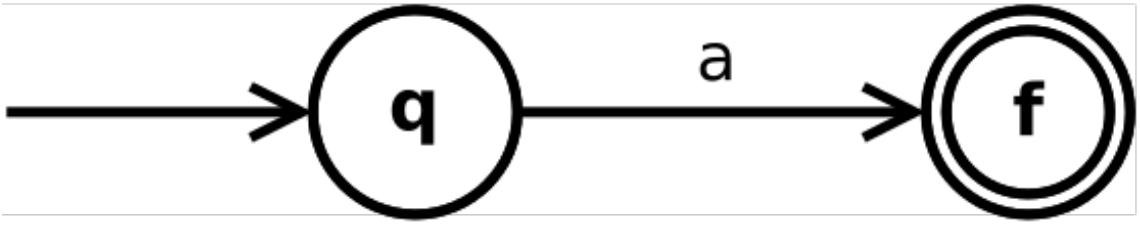

Figure 1: Thompson's construction - base case [6].

The operator "|" $(or)inaregularexpressionisrealisedasthefollowingconstructionthatcombinesthetu$

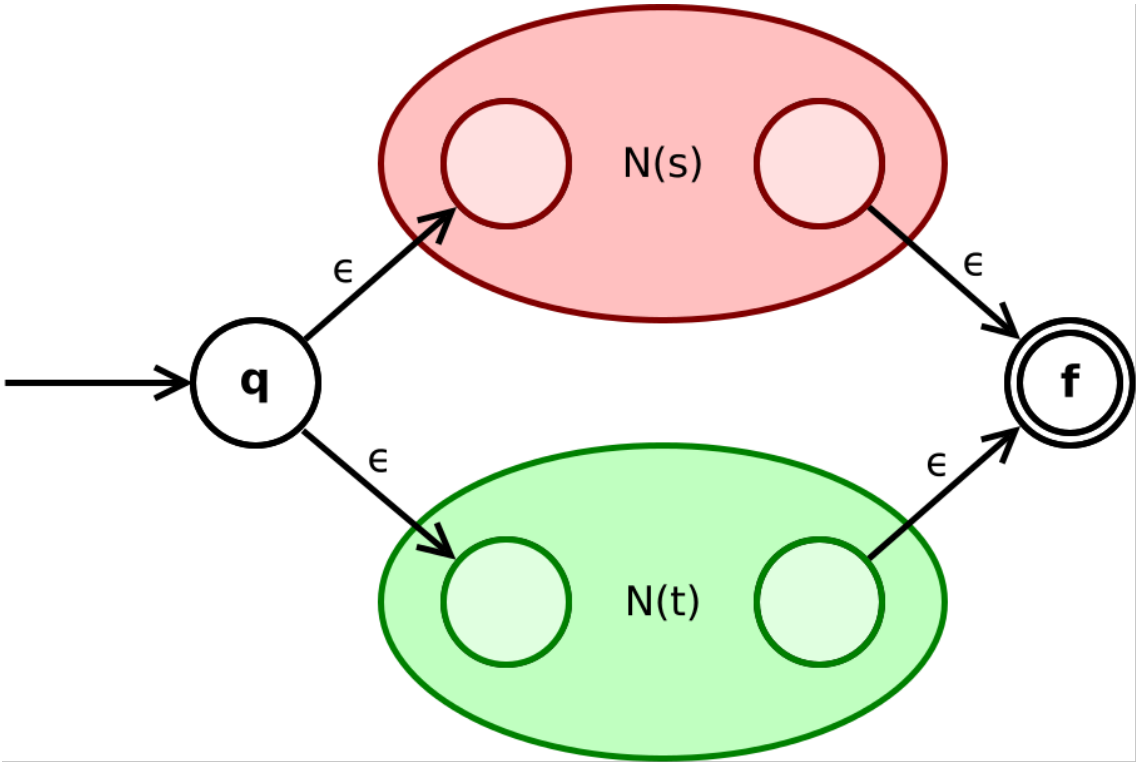

Figure 2: Thompson's construction - or operator [6].

The concatenation operator "." is realised by the following construction. But here it is to point out that often in regular expressions this operator is eliminated. In the upcoming sprints we are going to add the capability to not write all the "." explicitly but in the current version this is still necessary. The used construction is as follows:

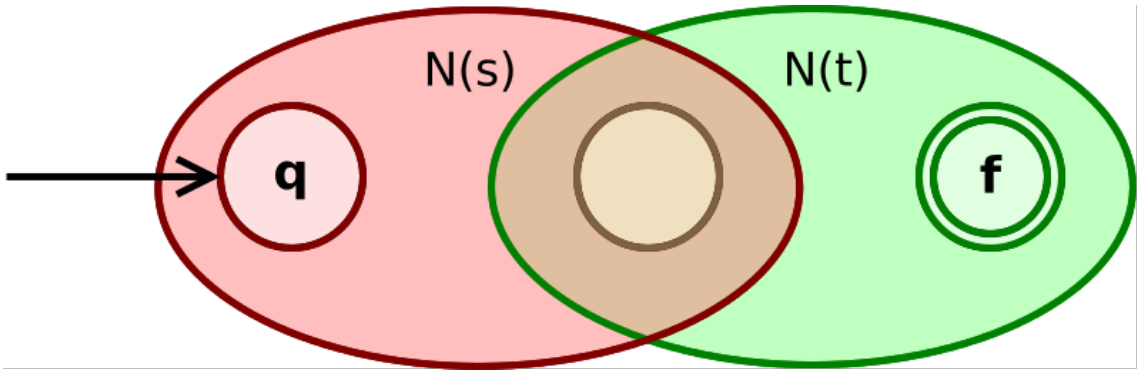

Figure 3: Thompson's construction - concatenation [6].

Finally the Kleene star operator "*" is constructed from a given sub nfa as follows:

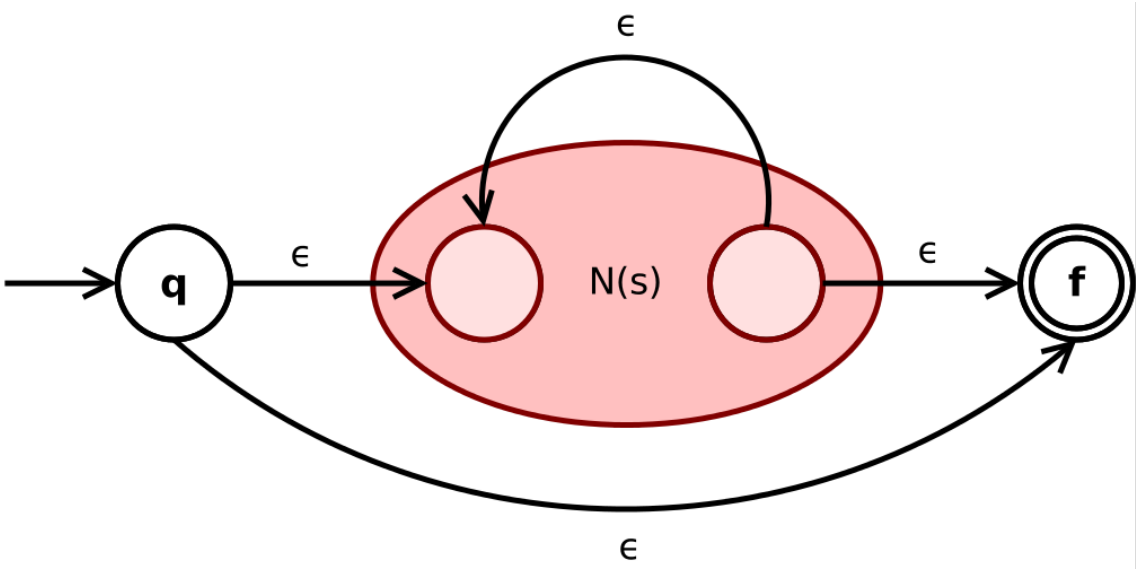

Figure 4: Thompson's construction - Kleene star [6].

[4].

2.2.3 Method checking whether a trace is fitting

We also managed to add the functionality to check whether a trace can be replayed on a given nfa model or not.

As a nfa is non-deterministic we decided to use a recursive approach to check for each possible move on the model whether this move can end in an accepting state. If one of those possibilities finds a accepted way then we return true. We decided to use depth first search because this tends to perform better when the search is far away from the start which is the case for event logs with a certain number of events.

2.3 Documentation

For documenting the code, we have used Pdoc. It is a software package for generating API documentation for Python language. Documentation is generated from the source code docstrings and HTML documentation is generated for chosen Python modules. The html file for sprint 1 is uploaded in the Git repository and can be accessed by going to *library - html - nfa.html*.

2.4 Testing

For this sprint we have done the manual testing. A few test cases can be seen in Figure 5 and Figure 6 below:

```
# Test section

myNFA = Nfa("TestNFA")
p1 = Place("Greating")
myNFA.add_place(p1, True)
p2 = Place("Start Small Talk")
myNFA.add_place(p2)
p3 = Place("End Small Talk")
myNFA.add_place(p3)
p4 = Place("Good Bye")
myNFA.add_place(p4, False, True)
t1 = Transition("a", p1, p2)
myNFA.add_Transition(t1)
t2 = Transition("b", p2, p2)
myNFA.add_Transition(t2)
t3 = Transition("b", p2, p3)
myNFA.add_Transition(t3)
t4 = Transition("c", p3, p4)
myNFA.add_Transition(t4)


print(myNFA.places)
print(myNFA.transitions)

print(myNFA.is_fitting(["a", "b", "c"])) #True
print(myNFA.is_fitting(["a", "b", "b", "b", "c"])) #True
print(myNFA.is_fitting(["a", "a", "b", "c"])) #False
print(myNFA.is_fitting(["a", "c"])) #False
print(myNFA.is_fitting(["a", "b"])) #False

myNFA.add_Transition(Transition(SpecialActivities.EPSILON, p2, p4))

print(myNFA.is_fitting(["a"])) #True

myRegexNfa = expression(["a", "*", "|", "(", "c", ".", "d", ")", "|", "(", "e", ".", "f", ")"])
myRegexNfa.print()

print("Regex: ")
print(myRegexNfa.is_fitting(["a", "a"])) #True
print(myRegexNfa.is_fitting(["a",])) #True
print(myRegexNfa.is_fitting([])) #True
print(myRegexNfa.is_fitting(["c", "d"])) #True
print(myRegexNfa.is_fitting(["e", "f"])) #True
print(myRegexNfa.is_fitting(["a", "c"])) #False
print(myRegexNfa.is_fitting(["a", "c", "d"])) #False
print(myRegexNfa.is_fitting(["x",])) #False
print(myRegexNfa.is_fitting(["c"])) #False
```

Figure 5: Test cases

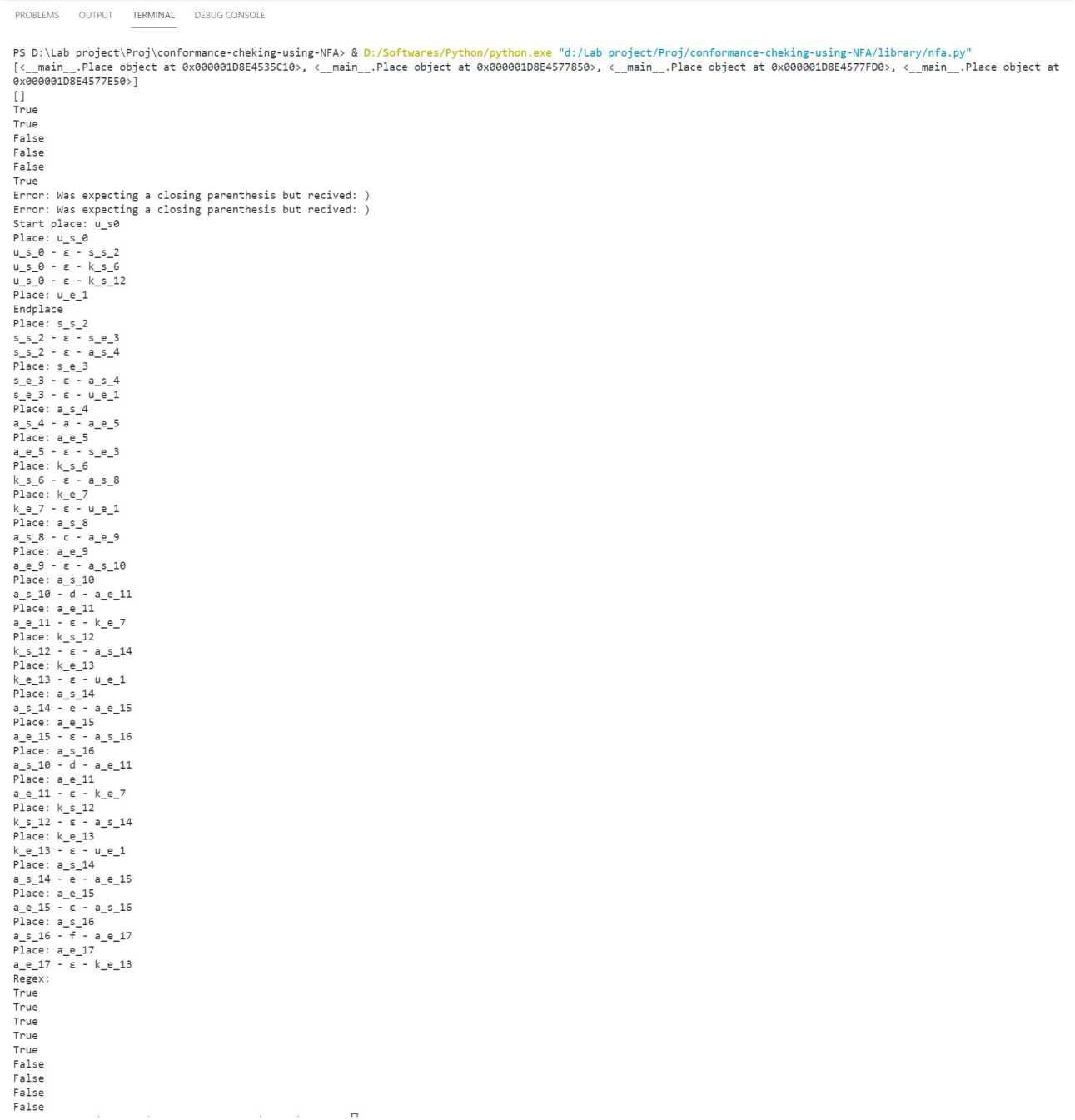

Figure 6: Test cases

3 Phase Review

3.1 Asad Tariq

I have found the whole team very energetic and ambitious in the sprint 1. I am very much impressed by the teamwork and honesty everyone has shown. We have met many times on online meetings and discussed the deliverable thoroughly. Everyone showed a great level of responsibility and have done their respective tasks in time. It was fun and very informative to have theoretical discussions regarding the development of NFA models. We got some extra time in this sprint and worked on another approach to model NFAs. Now we have a agreed upon approach to our implementation, we will be bringing in the Unit Testing module in the next sprint.

3.2 Mahmoud Emara

This first sprint was our first challenge in the implementation phase. I think we handled it well. We were able to achieve the goals we aimed for in this sprint. All the team members participated in the theoretical discussion and took their jobs seriously. The communication was the key in this sprint. We have not worked on Unit testing in this sprint as we decided to focus on the initial implementation and foundation of the code base. From next Sprint we will be focusing on Unit Test development in parallel to the development.

3.3 Syed Faizan Hassan

I am very happy and satisfied with the way every person owned their task and completed it according to deadlines we decided in our internal meetings. Even though its just the first official sprint we had the internal meeting thrice and I am very impressed that everyone took out the time to join all the meetings despite the busy routine of Regular semester. The meetings were very helpful as each one of us tried to pitch in their ideas and helped clear the misunderstandings regarding the requirements and task we are supposed to perform. The main focus of this sprint was to lay foundation of a strong code base for future development. We invested a lot of time in theoretical discussion and implemented the targeted features by discussing every thing in detail. Good thing was that we did not only focus on our part but we as a Team reviewed everyone's part so that we can help in-case anyone is facing difficulties in completing their respective part. Another good thing was that everyone worked actively on their tasks and maintained the dashboard in parallel. It was not needed to remind anyone to update their work on the Trello dashboard or on the overleaf document.

3.4 Lukas Liß

This sprint was the first sprint in which we started to code. This made this sprint especially hard in my point of view. We had problems to devide the coding task in part that can be developed by different people because most of the parts build up upon each other. Other tasks like creating the documentation could also only be done once there was a first implementation. That is why I think the first part of the sprint was a little bit more unstructured then the ones before. In addition this was the time were we discuvered many details in the implementation that we need to discuss as the early decisions in a project goes a long way so they have to be thought through well. In the middle of the sprint we had some meetings were we discussed very openly how we can improve the way we

work together in the rest of the first sprint but also for the rest of the project. We all saw similar ways to improve the team work, so we started to communicate more and in smaller iterations. I think it was also helpfull that we were more active in our WhatsApp group in the second half of the sprint as this drastically increased the speed in which we were able to solve problems. Finally we were able to find a way how to distribute the workload as good as possible. Therefore I am happy with the result of our first sprint. We archieved even a little bit more then we planned to archive. Although the first half of the sprint was not optimal I think it is a good sign that we overcome the initial problems and managed to find a good way to develop together as a team.

From the coding perspective I really enjoyed this sprint because it introduced me to interesting topics like the thompson construction and different parsing algorithms. During this early implementation phase I was able to test multiple different aproaches and compare them. I think that I was able to expand my algorithmic knowledge and I am looking forward to improve the current implementation further.

3.5 Mina Khalid

This was a bit challenging sprint as the scope was very vast and we had to come down to the starting point of execution leg keeping in view the dependencies of different modules. We had multiple sessions on the theoretical part and even came up with two different approaches within our group to go about it but eventually after quite a lot of discussions, finalized the one that appeared to be better. The team was in high spirits throughout and everyone did their best in coming up with solutions wherever anyone got stuck somewhere.

References

[1] Wil van der Aalst. Data Science in Action. Springer Berlin Heidelberg, 2016.

[2] Rabin, M. O. and Scott, D. Finite Automata and Their Decision Problems.IBM Journal of Research and Development, 1959, doi: 10.1147/rd.32.0114

[3] Alfred Vaino Aho; Monica S. Lam; Ravi Sethi; Jeffrey D. Ullman (2007). "3.7.4 Construction of an NFA from a Regular Expression" (print). Compilers : Principles, Techniques, Tools isbn: 9780321486813

[4] Burge, W.H. (1975). Recursive Programming Techniques. isbn: 0-201-14450-6

[5] R. Rastogi, P. Mondal and K. Agarwal, "An exhaustive review for infix to postfix conversion with applications and benefits," 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom), 2015, pp. 95-100.

[6] https://en.wikipedia.org/wiki/Thompson