MY TECHNICAL REPORT

GAUTIER AXEL

July 2024

Aesthetic Regular Expression Generation from Symbolic Finite Automata

NEWTON JIM agautier@lre.epita.fr

Abstract

This paper is about Symbolic Finite Automata and extracting Regular Expressions from a given Symbolic Finite Automata that we will call Sigma-DFAs. Those Sigma-DFAs represent Regular Type Expressions which we will call RTEs. We are trying to compare two Sigma-DFAs obtained by creating a random RTE, creating a Sigma-DFA from the RTE and then extract a RTE from the Sigma-DFA, we will then recreate another Sigma-DFA and compare both of them.

The problem we have with said extraction is that the algorithm we are using for extraction and generation, which is the state elimination algorithm, creates an exponentially large RTE when extracting which greatly slows down testing.

My objective here is to try to upgrade the extraction algorithm in order to obtain more Aesthetic Regular Type Expressions in order to make this testing faster and more accurate.

Contents

RTE: Regular Type Expresion, a regular expression whose language is made out of sets of Objects, for example Integers, objects of a certain class, or any sort of intersection or union between two sets
Sigma-DFA: Symbolic Deterministic Finite Automata representing an RTE, each of its transition holds a Set

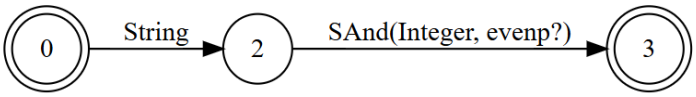
0.1 Introduction

The purpose of this paper is to try to improve one of our tests.
This test revolves around generating a random RTE, from which we will create a Sigma-DFA, from which we will then extract an RTE and then recreate a second Sigma-DFA which we will then compare to the original Sigma-DFA.
The problem we are having with this cycle is that the extracted RTE is exponentially greater than the original one, which makes it hard for us to compare the resulting Sigma-DFAs. Here we are trying to improve the RTE extraction algorithm which is the state elimination algorithm in order to obtain more aesthetic, simpler and smaller RTEs after extraction.
The current state of the state elimination algorithm is that the state which is to be eliminated next is selected by calculating the product of the number of incoming and outgoing transitions, and selecting the state with the highest result, this state is then eliminated.
My work has been going in a few directions in order to obtain a better algorithm, some of which have provided positive results and others who are still in

the testing phase.
I will first write in more detail about Sigma-DFAs and RTEs, then I will shortly describe the state elimination algorithm, after this I will go through each different step of the state splitting algorithm, starting by treating self-loops, followed by cycles, and then by parallel loops.
I will then provide an example of a positive result from my testing of the algorithm and end on the future work that should be done on this project.

0.2 State of the Art

The language used for our Sigma-DFAs and RTEs is made of Sets and Subsets using both native and custom classes of Scala. Those classes contain and are not limited to Integer, Abstract Classes and much more. We are also including predicates as Sets, which represent every element which validates a certain predicate, such as "the object is even".
We have observed through testing that in most cases that we are unable to verify that there exists no element that is both a member of a certain class and that also validates a predicate.
We also allow a number of operations to be performed on those sets which are the Union, Intersection and Kleene's Star.
Those Sets are then combined with the same operations and concatenation to create RTEs which are then used to generate DFAs.
Sigma-DFAs may contain on their transition any Set or combination of Sets amongst all the possibilities mentioned before, this includes as well the SEmpty Set and STop set which respectively represent an empty set containing no elements, and the Set containing every element.

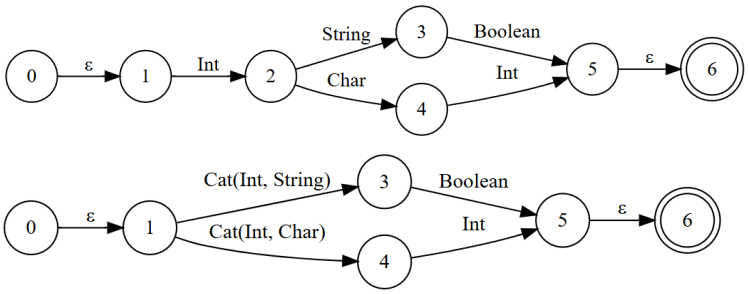


For example this Sigma DFA represents the Cat(Singleton(), SAnd(Singleton(Integer), SSatisfies(evenp)))

0.3 State Elimination Algorithm

The RTE extraction algorithm that was implemented at the start of my semester was the State Splitting algorithm, the way this algorithm works is that we add a new final and initial state and add a new Epsilon transition from each old final state to the new final state and a new Epsilon transition from each old initial state to the new initial state in order to only have one initial and one final state each.
Once this is done, we will proceed to eliminate every other state one by one, either selecting those at random or by following a certain rule.

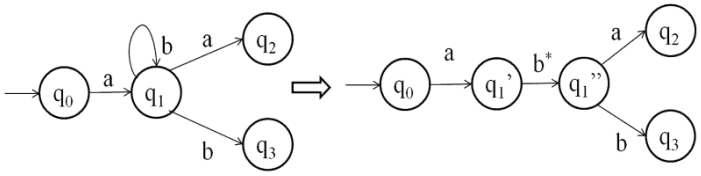
In our case we will select the state by calculating, for every state, the product of it's incoming transitions by it's outgoing transitions, and once this is done we will select the state with the highest product.
We will then connect every state from which there was an incoming transition to every state where there was an incoming transition with this specific state.
For example, considering the following Sigma-DFA, the Sigma-DFA we will obtain after eliminating the first state following this algorithm would be :



This step is to be repeated until every state other than the initial and final state that we have created earlier has been removed, we will then be left with only one transition between the initial and final state which will be the RTE which was represented by the Sigma-DFA.

0.4 State Splitting Algorithm

I will now describe the State Splitting algorithm as it was introduced in:
An elegant technique for obtaining shorter regular expressions, International Journal of Innovative Research Studies, DR. O. V Shanmuga Sundaram.
This paper describes an algorithm which suggests that splitting states with loops will result in obtaining shorter regular expressions when extracting the regular expression from an automaton.
The way this algorithm works is by going through every state and whenever a state with a loop is detected, we will create a new state, remove the loop, move every outgoing transition from the old state to the newly created one, and then create a transition whose label will now be Kleene's Star of the label of the loop.
The following image is an example of how the algorithm works on a state with a loop:

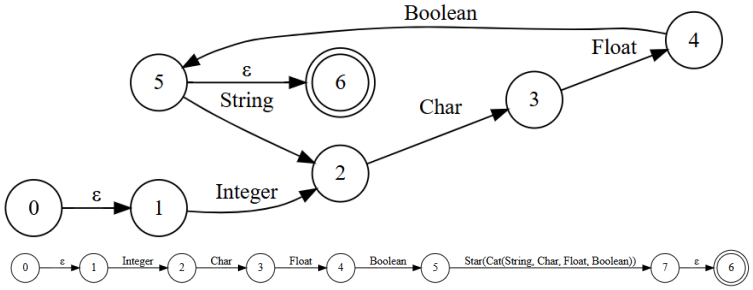


0.5 An Extension of the State Splitting Algorithm

In order to extend the State Splitting algorithm to the entirety of the graph, instead of reverting back to using the State Elimination algorithm once every self loop has been eliminated, I have thought of a way to extend the algorithm to every type of transition in order to be able to extract the smallest expression possible. To this end, I would follow the first step of the State Elimination algorithm which is to add a new Initial and Final state, and then follow a few steps.
The second step would be to eliminate every loop transition as described in the paper mentioned previously.
In the third step we will eliminate all cycles in a way described afterwards. And finally, in order to transform the graph into a general tree, we will get rid of all parallel paths, but in a way to make it easier to make them rejoin later, in order to optimise the regular expressions extracted. We are also using randomly generated Sigma-DFAs in our testing which might make some unusual transitions appear, which is why the algorithm needs to be able to treat every corner case possible.

0.5.1 Cycles and backward transitions

To follow our objective to transform our graph into a general tree we will now get rid of cycles, an easy example is to find which are the outgoing transitions of this cycle and then calculate what the regular expression is when starting at this state and returning to it, for example:

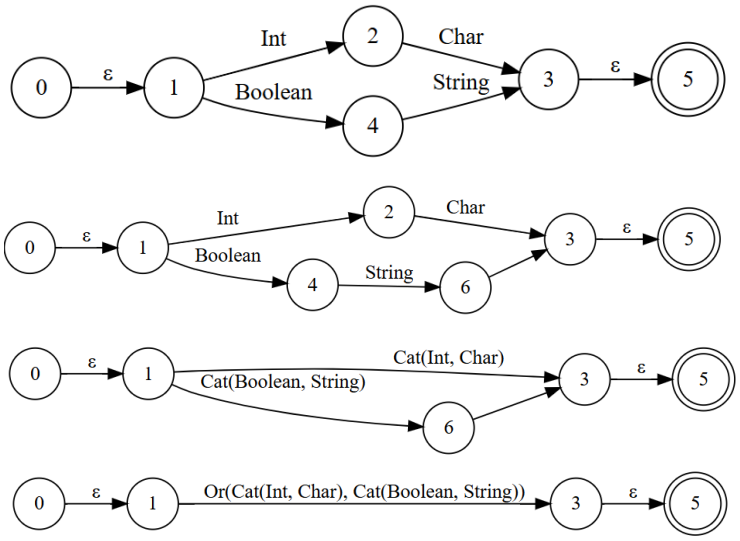


Here we have eliminated the backwards transition while not changing the language represented by the Sigma-DFA.

0.5.2 Parallel transitions

In order to continue transforming our graph into a general tree we shall now eliminate every parallel transitions, and replace them with symbolic transitions, these transitions will effectively work like Epsilon transitions but will make it easier for us to calculate the expression contained between two states As an

example:

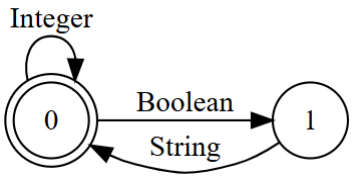


There is still a lot of testing and edge cases to do.

0.6 Conclusion

0.6.1 Results

First of all I would like to introduce a positive result that came from the implementation of this algorithm: Here is the original Sigma-DFA from which we will extract an RTE, the RTE used to generate this Sigma-DFA is Star(Or(Integer), Cat(Boolean, String)).



Here are the results we have obtained from each algorithm State Elimination Algorithm:
Or(Cat(Star(Int), Boolean, Star(Cat(String, Int, Boolean), String), Star(Int), Star(Int))

RTE extracted using the presented algorithm:

Star(Cat(Star(Integer), Star(Cat(Boolean, String))))

0.6.2 Future work

Most of the algorithm presented here still needs testing and some corner cases might not have been taken into account. Some comparisons should also be done as to whether it wouldn't be better to use the state elimination algorithm once the self loops have been eliminated or not.

Copying this document