Bachelor Thesis

# Evaluating Learning Algorithms for Regular Expressions

Lisa Busser

submitted at: 10.12.2018

**Technical University of Kaiserslautern**
**Embedded Systems Group**
**Department of Computer Science**

| | |
|---|---|
| Degree Programme: | Bachelor Applied Computer Science |
| Supervisor: | Prof. Dr. Klaus Schneider |
| Second Supervisor: | M.Sc. Marc Dahlem |

# Evaluating Learning Algorithms for Regular Expressions

## Zusammenfassung

Smartphones haben in nahezu jeden Lebensbereich Einzug gehalten. Zusammen mit der steigenden Verbreitung von Smartpones hat die Texterkennung aus Bildern immer mehr an Bedeutung gewonnen. Um Informationen aus diesen, sich dynamisch ändernden, Bildern zu extrahieren, is es hilfreich Eingabemuster zu lernen. Diese Eingabemuster können als reguläre Ausdrücke dargestellt werden.

Diese Arbeit beschäftigt sich damit, verschiedenen Strategien zu betrachten, wie reguläre Ausdrücke von echten Eingabedaten gelernt werden können. Dafür wurden die zwei Algorithmen Gold und RPNI untersucht, die in der Lage sind, contextsensitive Sprachen mittels deterministischer Automaten zu lernen. Zusätzlich gibt es einen Ausblick auf mögliche Verbesserungen dieser zwei Algorithmen. Sie werden gebenchmarked und miteinander verglichen, um heraus zu finden, wie gut die Ergebnisse die realen Eingabedaten wiederspiegeln abhängig von der Größe der Datensets.

# Abstract

Smartphones are included in nearly every area of life. Together with the increasing distribution of smartphones optical character recognition is of particular importance. To extract information out of this dynamic environment of different pictures learning petition patterns is helpful. This petition patterns can be displayed as a regular expression.

This thesis considers different strategies to learn regular expressions from a real world input sample and examines which is most convenient. Therefore Gold's algorithm and Regular Positive and Negative Inference (RPNI) have been investigated who are able to learn a context sensitive language via deterministic automatons. Additionally there will be an outlook about possibilities of further improvement. The two algorithms will be benchmarked and compared to each other how good the solutions of the algorithms will cover the real world problem with different sizes of the input sample.

# Eidesstattliche Erklärung

Ich erkläre an Eides Statt, dass ich die vorliegende Bachelor Thesis mit dem Titel *Evaluating Learning Algorithms for Regular Expressions* selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe und dass ich alle Stellen, die ich wörtlich oder sinngemäß aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe. Die Arbeit hat bisher in gleicher oder ähnlicher Form oder auszugsweise noch keiner Prüfungsbehörde vorgelegen.

Ich versichere, dass die eingereichte schriftliche Fassung der auf dem beigefügten Medium gespeicherten Fassung entspricht.

Kaiserslautern, den 10.12.2018

(Lisa Busser)

# Contents


**1. Introduction** **1**

**2. What means learning** **3**

**3. Related work** **4**
3.1. Artificial intelligence . . . . . 4
3.2. Evolutionary algorithms . . . . . 4
3.3. Automata based algorithms . . . . . 5
3.3.1. Deterministic vs. non-deterministic automatons . . . . . 5
3.3.2. Context sensitive vs. context free grammars . . . . . 6

**4. Implementation** **7**
4.1. DeLeTe2 and why this algorithm is not suitable for the problem . . 8
4.2. Tesseract and the user defined patterns . . . . . 9
4.3. Gold's algorithm . . . . . 9
4.3.1. Prefix Tree Acceptor . . . . . 10
4.3.2. Gold Observation table . . . . . 11
4.3.3. Gold fill holes . . . . . 16
4.3.4. Gold build automaton . . . . . 18
4.3.5. Further Optimisations of Gold . . . . . 20
4.4. RPNI . . . . . 21
4.4.1. Prefix Tree Acceptor . . . . . 21
4.4.2. RPNI Merge . . . . . 22
4.4.3. RPNI Fold . . . . . 23
4.4.4. RPNI Compatible . . . . . 25
4.4.5. RPNI Promote . . . . . 26
4.4.6. Example run . . . . . 27
4.4.7. Further Optimisation of RPNI . . . . . 32

**5. Benchmarks and Result** **33**
5.1. Benchmark . . . . . 33
5.1.1. Benchmark Gold . . . . . 33
5.1.2. Benchmark RPNI . . . . . 34
5.2. Results . . . . . 35

**6. Closure** **40**

**A. Appendix** **41**
A.1. Graph descriptions . . . . . 41
A.1.1. 1.000 example input sample . . . . . 41

**B. List of Figures** **59**

**C. References** **60**

# 1. Introduction

Smartphones are included in nearly every area of life. In 2017 around 81% of the German civilisation over the age of 14 uses a smart-phone [sta]. So it's an obvious step, to offer more and more services running on the smartphone. This is not just a credo for the entertainment sector, many people also wind up their affairs with the help of their phones. For example, if you have an app at your company and want to trail a specific object, like a weight at a weight manufacturer company, you have to type in the specific article number. Or for example if you are customer at an insurance company and you want some special informations about your bill, you could type in your invoice number into the app, to get the informations you were looking for.

Typing on the one hand is very error-prone on the other hand the usability is not very high. Cause every smartphone has at least one camera, it seems a good idea to not let the user type in the article number, but just let them take a photo of the target number or text and let the fields be filled in automatically. But also the optical character recognition (OCR) software, which parses the picture, is making mistakes. Or the picture can be that bad, that the recognised text doesn't make sense any more. So just taking a photo seems not to be the solution for getting a higher usability.

There is the need of a second instance, that controls the result of the optical character recognition, to improve the usability. This second instance should be able to check, if the result of the optical character recognition makes any sense for the given context, or if it's an impossible solution. In case of an impossible solution, it could just make an additional picture and run the OCR again to not bother the user with a result, which is now known to be incorrect. In case of article numbers that is a clear decision, because no company assigns article numbers on a random base. Normally there is a pattern how article numbers are build. It's the same for invoice numbers, addresses, IBAN numbers and many many more. So the second instance is testing if the solution of the OCR makes any sense. If not, it can initiate to take an additional picture and run the OCR again.

This thesis considers, how such a second instance could be build. The basic problem is, to build a regular expression, which describes the use case best. For example find the regular expression, which describes the building of all article numbers, used in that company. There the word learning comes into play. A general overview about what learning is, is described in section 2. In this thesis with learning is meant to get a finite set of examples as an input and create a infinite language as an output, that still corresponds to the input.

There are many different ways, to learn a regular expression. There will be explain some different ways, to solve that problem in chapter section 3. In this thesis, the focus is on how to learn a regular expression on the base of a positive

and a negative set of examples. The idea was, to start at a ground base of truth of positive examples. The negative examples are added over the time, because every time, the users scans something and it wasn't caped right, he will correct it. Out of that correction there is an additional positive and an additional negative example every time. To not waste the information contained in the negative examples, the algorithm should be able to use both kinds of information sets. Therefore Gold's algorithm and RPNI were chosen which are described isection 4. In this chapter, the difficulties during implementation and why the first choose, DeLeTe2 wasn't convenient for that specific use case, are explained. Additionally there is a small outlook how further optimisation can improve the algorithms.

In the chapter benchmarks and results section 5, there are some tests about the runtime of the two algorithms and the explanation of the results.

In the conclusion, there will be explained, if the strategies focused in this thesis, will be usable to improve the usability of OCR in applications.

# 2. What means learning

In linguistic usage learning is often connected to school or education. The acquirement of theoretical knowledge or the improvement of mobility of the human body [lea].

This form of learning is not considered in this thesis. When talking about learning, in this case it has nothing to with shaping abilities of any kind of biological creatures at all.

Here learning is in the context of algorithms and theoretical computer science. In strict sense learning is meant in the context of grammatical inference.

In context of grammatical inference when talking about learning means, to learn a language representation. This can be done by different kinds of input. Some learners are learning directly from texts, others from an example set of positive and negative examples, like the algorithms presented in this thesis. Some learn from a teacher or anything or somebody who provides any kind of knowledge about the elements contained in the target language.

One of the basic learning problems is to decide the **membership problem**. The target is, to decide if a given example is part of the language or not. At the end of this thesis in section 5 how the membership problem can be decided for the algorithms explained in section 4.

# 3. Related work

In this chapter, different ways to learn a regular expression are presented. Over the last 40 years there has been plenty of research about learning different types of languages. It turns out, that basically there are three different strategies to learn a language. A first option is to choose a nature inspired algorithm, especially an evolutionary algorithm. The second possibility is to use an artificial intelligence approach like learning via neuronal networks and the third opportunity is, to build the regular language with the help of automata based algorithms. Since the focus in this thesis is on automata based algorithms, there will only be a short introduction about nature inspired solutions. But all strategies have in common, that they use heuristics, because finding the optimal solution is NP-Complete [Gol].

## 3.1. Artificial intelligence

There had been some attempts to learn a regular expression via neuronal networks. In 1992 C.L. Giles published a paper with the title: 'Learning and Extracting Finite State Automata with Second-Order Recurrent Neural Networks' where he explained that he was able to learn small regular grammars. Therefore he uses a set of strings divided into a positive and a negative input sample [GMC+92]. A regular grammar is a type-3 grammar in Chomsky hierarchy [Cho56] and therefore the most simple type of grammar.

Geoff Hinton said, "I think that the most exciting areas over the next five years will be really understanding text and videos. I will be disappointed if in five years' time we do not have something that can watch a YouTube video and tell a story about what happened. In a few years time we will put [Deep Learning] on a chip that fits into someone's ear and have an English-decoding chip that's just like a real Babel fish." [Geo]

In [Man15] Christopher D. Manning named the deep learning approach a tsunami in Natural language processing. But additionally he said that computational linguistic must not be afraid.

At least it is hard up to impossible right now to give any guarantees with neuronal network approaches, so grammatical inference by learning with automatons right to exist.

## 3.2. Evolutionary algorithms

Evolutionary algorithms like genetic algorithms are of a special interest for grammatical inference, because in the field of bio-computing they are also dealing with languages and strings.

Genetic algorithms maintain a lot of strings. Each string encodes a given solution

to the learning problem. After running specific genetic operations, the population grows and evolves itself. Each improvement or new string must be accepted by a fitness function. In the special case of grammatical inference, there are some issues, that should be paid attention to.

First of all, the question is, what search space the algorithm is running in. Are only the correct examples part of it, or all. The next big problem is, how to find the string set of the first generation. The genetic functions, like the fitness function mentioned earlier must be defined. Typically there are some mutation functions. Each symbol of a string can mutate to a different string similar to DNA mutations. Additionally there must be a of crossing over operations. They mix up the information of two strings to produce different siblings for the next generation, like in real world, when a new child is born. The genetic information of both parents are mixed together. The crossing over operation did this for algorithms. A least the big question is, how to build the fitness function. This fitness function decides which new generated children will stay in that new generation. The problem here is, that there must be a comparison between two solutions and the question is, on which base that comparison should occur.

There are also many more details, which can change the result dramatical. For example the number of generations or how many strings should be kept in each generation.
To get further information about evolutionary algorithms, please take a look into [Hig10].

## 3.3. Automata based algorithms

A regular expression is the easiest representation form of a language for humans to understand. But only a few algorithms deal directly with these expressions. Even if the output is a regular expression, internally most of the algorithms are dealing with automatons. The algorithm presented by Efim Kinber in [Kin08] is one of the few algorithms directly dealing with regular expressions. But this algorithm is only able to learn regular languages which is a Typ-3 grammar in Chomsky hierarchy [Cho56] and therefore a very basic one. Algorithms for learning grammars of a higher Chomsky hierarchy level are mostly dealing with automatons.

### 3.3.1. Deterministic vs. non-deterministic automatons

Many algorithms, also for example the Gold algorithm, which is detailed discussed in section 4, have different opportunities to implement them. In this thesis, Golds algorithm is implemented with deterministic finite automatons. There is also an opportunity, to learn with Golds algorithm on the base of non-deterministic automatons but the complexity of implementation raises dramatically. De la

Higuera says, that just in favour for the much increased workload, the result for using non-deterministic automatons instead of deterministic ones isn't much better [Hig10]. So everyone must decide on his one, if the problem is worth to spend that much more resources for maybe just a little improvement of the result.

In this thesis, the decision was against non-deterministic automatons, because for testing if Gold's algorithm and RPNI are helpful to improve usability in Optical character recognition, small deviations are not that important.

### 3.3.2. Context sensitive vs. context free grammars

A context free grammar is a Type-2 grammar in Chomsky hierarchy. Context sensitive grammars are Type-1 grammars and therefore include all Type-2 grammars [Cho56].
Since this thesis is focused on learning regular expressions that describe real world constructs, like IBANs or invoice numbers, learning context sensitive grammars is more interesting. Both algorithms described in section 4, Gold and RPNI are able to learn context sensitive grammars. Yasubumi Sakakibara explains an easier algorithm, to learn the more restricted context free grammars [Sak92]. THis is just one way to learn a context free grammar, of course there exists many more.

# 4. Implementation

The background of this thesis was to improve the work flow in a mass metrology labour. They are working very closely at the borders that are given from physics. For example, the weights are made of steel. On account of moving them through the earth magnet field they slowly magnetize. The comparators used in mass metrology are able to measure the differences of orientation of the weight due to the magnetisation. In such a high end labour, change in temperature is a big problem because the comparators are influenced by the temperature and if it changes more than 1 degree over 4 hours, all measurements must be taken again. So in the near future tablets will be introduced. That frontend of the software displays all the values from the climate stations and comparators about the current running weighting procedure. In the past and still today, that values are displayed on laptops. That reduces the emission heat immense, and make it easier to pass the maximum temperature deviation so that viewer measurements have to be repeated. [MAR]

The basic reason for introducing tablets was the temperature emission but it turned out to be a good opportunity to also improve the work flow and increase the usability of the software. Every weight must be tracked. There is the possibility to follow the trend of a single weight over years and therefore it must be recognizable. So the weights have a number on them, which is laserd on them by a specific technique that avoids loosing or adding material during the process. At the beginning of each measurement the worker has to type in the number of the weights he measures. Typing in the value often leads to typos or other mistakes. That could be avoided and since the number on the weights are not randomly distributed learning a regular expression should also be possible, like it is for IBAN's ore invoice numbers. And since every tablet has at least one camera why not making a photo and parse the number from the photo.

Parsing the photo to a string is not part of this thesis, so the idea was to use tesseract. Tesseract is an open source optical character recognition software on GitHub. The idea was, to make a photo of the weight and Tesseract parses the number on the weight to a string. The regular expression can be deposited directly in Tesseract. These so called 'user defined functions' are able to improve the recognition by helping to decide if the recognised pattern is valid or not. There have been some problems during implementation using Tesseract which are explained in subsection 4.2.

The algorithm that seems fitting best for learning the numbers on the weights was DeLeTe2 [DLT04]. Since there will be positive and negative examples due to the fact that a correction of a wrongly taken string will always lead to a new positive and a new negative example an algorithm, which can improve his result by the help of both input sets, seems very effective. Additionally DeLeTe2 is an

iterative learner, so it seems a good choice for integrating the new examples, that are always taken after a new recognition. Why DeLeTe2 was at the end not suitable for the problem is explained in subsection 4.1.

Since DeLeTe2 wasn't working, an new algorithms had to be found. The choice came to Gold's algorithm, which is a very basic one and explained in subsection 4.3. Additionally, the algorithm RPNI cam (Regular Positive and Negative Inference)e into focus and the question came up, which of the two of them might fit better for solving the given problem. RPNI is explained in subsection 4.4. The decision which algorithm seems to be better appropriated is discussed in

## 4.1. DeLeTe2 and why this algorithm is not suitable for the problem

The first idea was to learn regular expressions with the Algorithm DeLeTe2 [DLT04]. At first sight, it seems to be the best choice for the given problem. DeLeTe2 is able to learn random languages, on the base of residual finite automatons (RFSA). The size of RFSA's is smaller than the size of deterministic finite automatons(DFA), so it is an advantage against algorithms using DFA's because the resulting automaton and therefore the resulting regular expression is smaller. Also it processes positive and negative examples, which is much better than algorithms, which only processes positive examples, because he doesn't waste the information contained in the negative ones. Additionally DeLeTe2 is able to learn iteratively, so getting new information, which is happening all the time in the given use case doesn't mean to start again from zero but to improve the last result.

Since in the use case, where the employee is takes a picture of the target structure and the OCR processes the photo the result is one additional information, which might help to improve the result. If the OCR caps the structure wrong, than the correction of the user is an additional information, which can improve the regular expression. So either there is one or two information, after each newly taken photo. Due to economise resources the algorithm DeLeTe2 would be perfect, because it doesn't has to start from zero again, and do the whole computation with an increased dataset. But it can start from the regular expression and therefore from the automaton, it calculates the step before and just improve this result by the help of the new examples.

But DeLeTe2 like it was published in [DLT04] may not be consistent [GRC]. After DeLeTe was published, the author Denis et. al. reported, that there is a consistent version of DeLeTe2, but he didn't explain, how he did that.

That inconsistency lead to not use the algorithm at all, because not being able to guarantee that the result of one step is correct, is a really big problem at an iterative algorithm.

## 4.2. Tesseract and the user defined patterns

Tesseract provides a function, that there is a possibility to implement user defined functions [Tes]. These user defined functions are an own defined version of regular expressions. Own defined, because they use a different syntax and semantic than normal regular expressions. This includes, that there is no possibility to give different alternatives in one expression. So each expression defines only one specific pattern, which is acceptable. That isn't quite nice for the regular expressions found in this Thesis, because at the end the solution must be transferred into the semantic schema of Tesseract. It turned out that Tesseract no longer supports such user defined patterns at all due to an open bug, which seems not to be relevant for many people as it is open since 30th of august in 2016 [bug].

Since fixing this bug would overshoot the volume of this thesis, there is just a theoretical approach, if the presented algorithms might help to improve the usability of OCR and are beneficial in order to the use case.

## 4.3. Gold's algorithm

The Gold algorithm is part of the so called informed learners. That means, that he gets his complete input sample at the beginning of the algorithm. This input sample contains a set of positive examples, which are part of the target language and a set of negative examples, which are not part of the language. The algorithm was published by E. Mark Gold which is the reason why this algorithm is known as Gold algorithm. The Gold algorithm especially in the implementation explained in this thesis is a very basic algorithm in the field of learning a regular expression from positive and negative input samples. But there are some algorithms, which take Gold as a base and optimize from there. On example of an improved version of Gold is DeLeTe2 but since this algorithm is not suitable for the problem maybe its derivation is.

The Gold algorithm contains 4 main steps:

1. Build the prefix tree acceptor (subsubsection 4.3.1)

2. Calculate the $RED$ and the $BLUE$ nodes which describe if a prefix contains a new information or if it is already contained in the other states of the automaton which leads to an observation table (subsubsection 4.3.2)

3. Guess all unknown states in the observation table (subsubsection 4.3.3)

4. Build the resulting automaton (subsubsection 4.3.4)

### 4.3.1. Prefix Tree Acceptor

The prefix tree acceptor (PTA) is deterministic finite automaton, which is tree like. This automaton is defined as $\text{PTA}(\langle S_+, S_- \rangle) = \langle \Sigma, Q, q_\lambda, \mathbb{F}_\mathbb{A}, \mathbb{F}_\mathbb{R}, \delta \rangle$, where $\Sigma$ is the alphabet, $Q$ is the set of states in the automaton, $q_\lambda$ is the starting point, $\mathbb{F}_\mathbb{A}$ is the set of accepting states, $\mathbb{F}_\mathbb{R}$ is the set of rejecting states and $\delta$ is the set of transitions.
The sets of positive and negative examples are provided by the informant. During the further explanation $S_+$ is the set of positive examples and $S_-$ is the set of negative examples, such that $S_+ \cap S_- = \emptyset$.
A sample $S = \langle S_+, S_- \rangle$ is called **weakly consistent** $if_{def}$ $\forall x \in S_+, \delta(q_\lambda, x) \in \mathbb{F}_\mathbb{A}$ and $\forall x \in S_-, \delta(q_\lambda, x) \notin \mathbb{F}_\mathbb{A}$.
A sample $S = \langle S_+, S_- \rangle$ is called **strongly consistent** $if_{def}$ $\forall x \in S_+, \delta(q_\lambda, x) \in \mathbb{F}_\mathbb{A}$ and $\forall x \in S_-, \delta(q_\lambda, x) \in \mathbb{F}_\mathbb{R}$.
A **prefix closed set** is a set of strings $S$ where $uv \in S \Rightarrow u \in S$. [Hig10]

The prefix tree is strongly consistent and prefix closed. With the help of the example set

$$S_+ = \{a, abb, bba, bbb\} \tag{1}$$
$$S_- = \{b, aa, bb, bab\} \tag{2}$$

the following steps are explained.

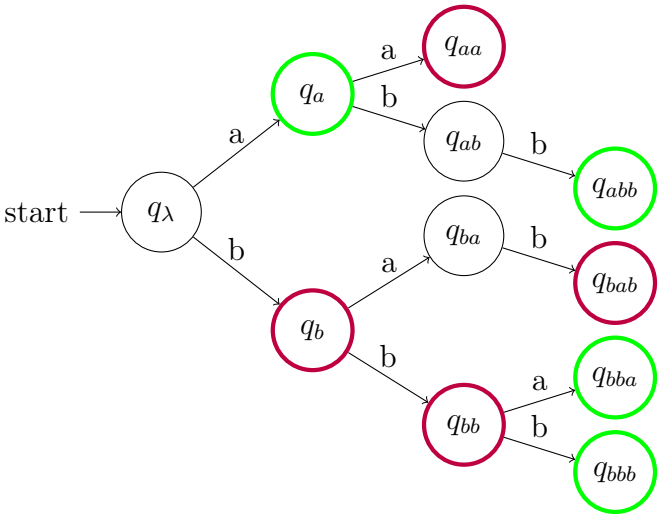


Figure 1: Prefix Tree Acceptor Gold

Input: $S_+$, $S_-$
Output: PTA

1. Set $q_\lambda$ as initial state
2. $\mathbb{F}_\mathbb{A} = \emptyset$
3. $\mathbb{F}_\mathbb{A} = \emptyset$
4. $Q$ is the set of prefixes of $S_+ \cup S_-$
5. For each state $q_{ua}$ add the transition from $q_u$ to $q_{ua}$ via $a$ to $\delta$
6. For each $q_u \in Q$ do:
   a) If $q_u \in S_+$ then $\mathbb{F}_\mathbb{A} = \mathbb{F}_\mathbb{A} \cup q_u$
   b) If $q_u \in S_-$ then $\mathbb{F}_\mathbb{R} = \mathbb{F}_\mathbb{R} \cup q_u$
7. If a state $q_u$ would be both element of $\mathbb{F}_\mathbb{A}$ and $\mathbb{F}_\mathbb{R}$, the given set of examples is inconsistent and therefore there is no regular language describing the set of examples

Figure 2: Build prefix tree acceptor

In Figure 1 the prefix tree acceptor of $S_+$ and $S_-$ is shown. This is the starting point for running the Gold algorithm.
The pseudocode for building the prefix tree acceptor is shown in Figure 2.

In this graphics and in all other automatons drawn in this thesis, the states with the red circle around them are rejecting, and the states with the green circles around them are accepting. The states with black circles are not defined in either the accepting or rejecting set of states, so they are unknown. These unknown states are justified in the fact, that there is a state in the prefix tree acceptor for every prefix of the input sample but not each prefix is part of the accepting or rejecting input samples. The states which aren't a member of one of those two sets are unknown.

For getting a detailed look into the implementation, please take a closer look into the source code.

### 4.3.2. Gold Observation table

The next step in learning a regular language, described by the example set, is building a observation table. This table shows each information about the connection of two states. Therefore, there must be introduced a colouring of the states.

A $RED$ state means, this states adds really a new information to the resulting automaton and therefore must be in the resulting set of states. A $RED$ state can't be merged to an other $RED$ state. A $BLUE$ state, is the child of a $RED$ state. This state might be a new information. Therefore it must be checked, if the $BLUE$ state can be merged to a $RED$ state. Merging two states means, one of the state isn't really a new information and therefore doesn't has to be a member of the resulting automaton. Since the clue is to get a minimalistic result as possible, states which don't have to be a member of the automaton aren't added. If a $BLUE$ state contains a real new information, it is promoted to the set of $RED$ states. The child's of the promoted state, ordered by the prefix tree acceptor, are now promoted to be $BLUE$ states. If no blue state contains any new informations, the final sets of blue and red states are found. In this chapter it is explained how to build the observation table step by step, and how the entries are filled in.
At first, the experimental set must be found. This set is a suffix closed list of $S_+ \cup S_-$.

A **suffix closed set** is a set of strings $S$ where $uv \in S \Rightarrow v \in S$.
The experimental set for the input sample $S_+, S_-$ is:

$$exp = \{\lambda, a, b, aa, ab, ba, bb, abb, bab, bba, bbb\}$$

Then the observation table for $RED = \{\lambda\}$ is build Figure 3.

| | $\lambda$ | a | b | ba | aa | bb | ab | bba | abb | bbb | bab |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lambda$ | $*$ | 1 | 0 | $*$ | 0 | 0 | $*$ | 1 | 1 | 1 | 0 |
| $a$ | 1 | 0 | $*$ | $*$ | $*$ | 1 | $*$ | $*$ | $*$ | $*$ | $*$ |
| $b$ | 0 | $*$ | 0 | 1 | $*$ | 1 | 0 | $*$ | $*$ | $*$ | $*$ |

Figure 3: Observation table $RED = \{\lambda\}$

In the first column, there are the $RED$ states and the $BLUE$ ones. The $BLUE$ states, are $a, b$ since they are the child's of $\lambda$, in the prefix tree acceptor, which is the first $RED$ state. In the describing row, there is the experimental set build earlier. The entries of the observation table are build like shown in Figure 4.

Exemplary for the $BLUE$ state 'a'. The first element in the experimental set is '$\lambda$'. So 'a' concatenated with '$\lambda$' is 'a' and 'a' is element of $S_+$ so the Observation table at [a][$\lambda$] must be 1. The second element in the experimental set is 'a'. 'a' concatenated with 'a' is 'aa' and 'aa' is an element of $S_-$ so the observation table at [a][a] must be 0. The third element in the experimental set is 'b'. 'a' concatenated with 'b' is 'ab' and 'ab' is not an element of $S_+$ as well as $S_-$ so the observation table at [a][b] must be $*$. All other values in the observation table are filled in analogously.

Input: Observation table
Output: Observation table

1. For each p $\in RED \cup BLUE$ which is in this first iteration $\lambda, a, b$
   a) For each e $\in$ experimental set
      i. If pe $\in S_+$ $\rightarrow$observation table [p][e] = 1
      ii. Else if pe $\in S_-$ $\rightarrow$observation table [p][e] = 0
      iii. Else observation table [p][e] = $*$

Figure 4: Fill in observation table

After this first observation table is filled, the next step is to search for new information. Therefore, the $BLUE$ states are of interest. So each row of a $BLUE$ state has to be checked, if it is contradicting to all $RED$ states.

A whole **row** of the observation table of state $u$ is refered as OT[u].
Two two rows OT[$u$] and OT[$v$] are **compatible** $\nexists e \in exp | OT[u][e] = 1$ and $OT[v][e] = 0$ or vice versa. If the two rows are compatible they are written as $u \backsimeq_{OT} v$.
The two rows OT[$u$] and OT[$v$] are **obviously different** if $\exists e \in exp | OT[u][e]$, $OT[v][e] \in \{0, 1\}$ and $OT[u][e] \neq OT[v][e]$. [Hig10]

For example [a][bb] = 1 and [$\lambda$][bb] = 0. So the $BLUE$ state 'a' adds really a new information to the set of $RED$ nodes, since it is obviously different to all existing $RED$ nods. Because of that, 'a' is promoted to the set of $RED$ states. Therefore, also the childs of 'a' which are 'aa' and 'ab' have to be promoted to the set of $BLUE$ states, since their parent is now $RED$. Now $RED = \{\lambda, a\}$ and $Blue = \{b, aa, ab\}$.

| | $\lambda$ | a | b | ba | aa | bb | ab | bba | abb | bbb | bab |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lambda$ | $*$ | 1 | 0 | $*$ | 0 | 0 | $*$ | 1 | 1 | 1 | 0 |
| $a$ | 1 | 0 | $*$ | $*$ | $*$ | 1 | $*$ | $*$ | $*$ | $*$ | $*$ |
| $b$ | 0 | $*$ | 0 | 1 | $*$ | 1 | 0 | $*$ | $*$ | $*$ | $*$ |
| $aa$ | $*$ | $*$ | 1 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |
| $ab$ | 0 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |

Figure 5: Observation table $RED = \{\lambda, a\}$

It is important, after promoting one $BLUE$ state to $RED$, to directly build a new observation table. Because, if first all $BLUE$ states which are inconsistent to all $RED$ states are found and then promoted together, there is no check if the additionally promoted $BLUE$ states are maybe compatible to each other. To avoid that problem and as a result maybe adding unnecessary states to the resulting DFA, each $BLUE$ state is promoted individually. After promoting 'a' to $RED$ the set of $BLUE$ states are controlled, if there is an additional one, which is obviously different to all $RED$ states. The $BLUE$ state 'b' is 0 at [b][$\lambda$] but 'a' is 1 at [a][$\lambda$], so 'b' is obviously different to 'a'. The observation table at [b][bb] = 1 but [$\lambda$][bb] = 0, so 'b' is also obviously different to $\lambda$ and therefore different to all $RED$ states (Figure 5). So now 'b' has to be promoted to the $RED$ states and the children of 'b' due to the prefix tree acceptor, which are the states 'ba' and 'bb', are promoted to the $BLUE$ states (Figure 6).

| | $\lambda$ | a | b | ba | aa | bb | ab | bba | abb | bbb | bab |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lambda$ | $*$ | 1 | 0 | $*$ | 0 | 0 | $*$ | 1 | 1 | 1 | 0 |
| $a$ | 1 | 0 | $*$ | $*$ | $*$ | 1 | $*$ | $*$ | $*$ | $*$ | $*$ |
| $b$ | 0 | $*$ | 0 | 1 | $*$ | 1 | 0 | $*$ | $*$ | $*$ | $*$ |
| $aa$ | $*$ | $*$ | 1 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |
| $ab$ | 0 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |
| $ba$ | $*$ | $*$ | 0 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |
| $bb$ | 0 | 1 | 1 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |

Figure 6: Observation table $RED = \{\lambda, a, b\}$

Finally, after promotion of 'b', only 'bb' is promotable, which leads to the final Observation table Figure 7. This resulting observation table is called **closed**, because $\forall u \in BLUE, \exists p \in RED : OT[u] = [p]$. [Hig10]

| | $\lambda$ | a | b | ba | aa | bb | ab | bba | abb | bbb | bab |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lambda$ | $*$ | 1 | 0 | $*$ | 0 | 0 | $*$ | 1 | 1 | 1 | 0 |
| $a$ | 1 | 0 | $*$ | $*$ | $*$ | 1 | $*$ | $*$ | $*$ | $*$ | $*$ |
| $b$ | 0 | $*$ | 0 | 1 | $*$ | 1 | 0 | $*$ | $*$ | $*$ | $*$ |
| $bb$ | 0 | 1 | 1 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |
| $aa$ | $*$ | $*$ | 1 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |
| $ab$ | 0 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |
| $ba$ | $*$ | $*$ | 0 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |
| $bba$ | 1 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |
| $bbb$ | 1 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |

Figure 7: Observation table $RED = \{\lambda, a, b, bb\}$

1. For all b $\in$ BLUE
   a) If $\exists r \in RED \mid b \trianglelefteq_{OT} rb$ then
      i. For all e $\in$ experimental set
         A. If OT[b][e] $\neq *$
         B. Then OT[r][e] $\leftarrow$ OT[b][e]
   b) Else return failed
2. For r $\in$ RED
   a) For e $\in$ experimental set
      i. If OT[r][e] $= *$
      ii. Then OT[r][e] $\leftarrow 1$
3. For b $\in$ BLUE
   a) If $\exists r \in RED \mid b \trianglelefteq_{OT} r$
      i. For e $\in$ experimental set
         A. If OT[b][e] $= *$
         B. Then OT[b][e] $\leftarrow$ OT[r][e]

Figure 8: Fill holes

### 4.3.3. Gold fill holes

The task of Gold fill holes is, to eliminate the $*$ in the observation table. These $*$ are called **holes** and represent a missing information. If there are no holes in the observation table, it is called **complete**. [Hig10]

The first step of Gold fill holes is to integrate additional information, contained in the $BLUE$ states, to the $RED$ ones. Therefore, each $BLUE$ state is taken into account one after another. Exemplary the first $BLUE$ state 'aa' is taken and compared to one $RED$ state after another. 'aa' $\backsimeq_{OT}$ 'a', and OT[aa] is only at OT[aa][b] different from $*$. The Observation table of OT[a][b] $= *$, so OT[aa][b] $=$ 1 is written into OT[a][b] which now is also 1. This is done for each $BLUE$ state until they are compatible to a $RED$ one. If a match is found, the next $BLUE$ state is focused on. This results is the new observation table Figure 9. This is a strategy called 'greedy'. There are also more intelligent ways to extract additional informations contained in the $BLUE$ states. One other, more elegant solution, is discussed in subsubsection 4.3.5.

| | $\lambda$ | a | b | ba | aa | bb | ab | bba | abb | bbb | bab |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lambda$ | 0 | 1 | 0 | $*$ | 0 | 0 | $*$ | 1 | 1 | 1 | 0 |
| $a$ | 1 | 0 | 1 | $*$ | $*$ | 1 | $*$ | $*$ | $*$ | $*$ | $*$ |
| $b$ | 0 | $*$ | 0 | 1 | $*$ | 1 | 0 | $*$ | $*$ | $*$ | $*$ |
| $bb$ | 0 | 1 | 1 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |
| $aa$ | $*$ | $*$ | 1 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |
| $ab$ | 0 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |
| $ba$ | $*$ | $*$ | 0 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |
| $bba$ | 1 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |
| $bbb$ | 1 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |

Figure 9: Observation table after first step of fill holes

The second step of fill holes changes all '$*$' in the $RED$ states to 1. This is a very optimistic estimation of the target language and many inputs might be accepting, although they are not. There is still a feedback that the states of the negative input sample will also be negative in the resulting automaton (subsubsection 4.3.4). But later on in the application a lot of input strings will be accepted, because of the generalization being so optimistic. That is the problem resulting out of all this unknown states in the observation table. The generalization, that is necessary to sample a language out of examples, will always lead to an abstraction which allows wrong decisions. Only an example which describes the language completely can avoid such wrong decisions. The examples, in the use case of this thesis are

given on a random base so having a complete example is very unlikely. Since getting such a complete example will probably stay a wish, any form of guessing the right language must be done. Just setting everything to true is a very optimistic estimation, but a possible one and for the given use case also not the worst one. This is because if the results of the OCR are often rejected, the usability wouldn't increase, because of long delays. So better to accept a false input and provide a fast result, than needing a lot of time to accept a false input. In the second case, the user would probably not use that feature, because it is faster for him to copy the input on his own.

The third step of Gold fill holes is to eliminate the $*$ in the $BLUE$ states. That is done analogously to the first step of Gold fill holes (Figure 11). This observation table is complete.

| | $\lambda$ | a | b | ba | aa | bb | ab | bba | abb | bbb | bab |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lambda$ | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| $a$ | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $b$ | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| $bb$ | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $aa$ | $*$ | $*$ | 1 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |
| $ab$ | 0 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |
| $ba$ | $*$ | $*$ | 0 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |
| $bba$ | 1 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |
| $bbb$ | 1 | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ | $*$ |

Figure 10: Observation table after second step of fill holes

| | $\lambda$ | a | b | ba | aa | bb | ab | bba | abb | bbb | bab |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lambda$ | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| $a$ | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $b$ | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| $bb$ | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $aa$ | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| $ab$ | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $ba$ | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| $bba$ | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $bbb$ | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Figure 11: Observation table after last step of fill holes

### 4.3.4. Gold build automaton

The last step of running one iteration of Gold's algorithm is to build the resulting deterministic finite automaton(DFA) and verify if the result is still corresponding to the input sample. Only states contained in $RED$ are states in the DFA. So when building the DFA, the first step is to build the set of states $q$. After that, the decision if the state belongs to the accepting states $\mathbb{F}_{\mathbb{A}}$ or to the rejecting states $\mathbb{F}_{\mathbb{R}}$. Therefore all $RED$ states are focused one after another. Fore each $RED$ sate $q_r$, all elements of the experimental set 'e' are considered. If there exists a state $q_{re}$, the concatenation of state r and experiment e, in the set of states $Q$ the state $q_{re}$ is accepted if OT[r][e]= 1 and added to the set of accepting states $\mathbb{F}_{\mathbb{A}}$, otherwise $q_{re}$ is rejected and added to $\mathbb{F}_{\mathbb{R}}$.

At least the transitions $\delta$ are build. To clear the understanding of how the transitions are build easier, a transition table is used (Figure 12). The columns are indexed by the alphabet $\Sigma$ of the input sample. $\Sigma$ contains all characters which are part of the input sample. In this example $\Sigma = \{a, b\}$. The rows of the transition table(TT) are indexed by the states of the automaton $Q$. The TT[$q_a$][a] = $q_\lambda$, which means, that there is a transition from $q_a$ to $q_\lambda$ via a.

The transitions of the transition table are build as follows. At first run over all $q_s \in Q$ and then in that loop for the second time over each $q_e \in Q$. The first $q_s$ is always the starting point of the transition. $q_e$ is the end point of the transition. Then iterate over each $\alpha \in \Sigma$. This $\alpha$ is the index of the transition. At least go over each e $\in$ experimental set. If OT[$q_s\alpha$][e] = OT[$q_e$][e] for each e, than there is a transition from $q_s$ to $q_e$ via $\alpha$ (subsubsection 4.3.4).

After running Gold build automaton Figure 14 is the resulting automaton. The definition of the resulting automaton $A$ $A = \langle \Sigma, Q, q_\lambda, \mathbb{F}_{\mathbb{A}}, \mathbb{F}_{\mathbb{R}}, \delta \rangle$. $\Sigma$ is $\{a, b\}$, which is still the same like it was at the PTA because the input samples haven't changed. $Q = \{q_\lambda, q_a, q_b, q_{bb}\}$ which is the set of states. These are of cause only a subset of the initial set of states, since the target was to generalize and therefore minimize the PTA. So having viewer states than before was a goal. $q_\lambda$ is the initial state, which is the same like before. $\mathbb{F}_{\mathbb{A}}$ is the set of accepting states which is now $\mathbb{F}_{\mathbb{A}} = \{q_a\}$. $\mathbb{F}_{\mathbb{R}} = \{q_\lambda, q_b, q_{bb}\}$ is the final set of rejecting states. And finally $\delta$ is the set of transitions.

It is interesting that after running the Gold algorithm each state in $Q$ has as many outgoing transitions as the size of $\Sigma$. So for every value in the alphabet there is an outgoing transition at each state.

The last step that has to be done, is to prove that the resulting automaton still corresponds to the input sample. Therefore, each example of the positive and the negative input sets has to be parsed by the automaton. If a example of the positive input sets ends at a rejecting or unknown state or an example of the negative input set ends at a accepting state the automaton is not corresponding to the input set so

the algorithm returns the prefix tree acceptor only made by the positive examples. The prefix tree acceptor of the positive examples always fits to the input example, since it parses the positive examples and non of the negative. But this result hasn't any generalisation at all, which makes the result as pessimistic as possible, because nothing else than the positive examples are accepted.

The result of the example input set is accepted, so the generalisation has nothing done, that is in conflict with the input sample.

| | a | b |
|---|---|---|
| $q_\lambda$ | $q_a$ | $q_b$ |
| $q_a$ | $q_\lambda$ | $q_a$ |
| $q_b$ | $q_\lambda$ | $q_{bb}$ |
| $q_{bb}$ | $q_a$ | $q_a$ |

Figure 12: Transition matrix

Input: complete observation table
Output: DFA $A = (\Sigma, Q, q_\lambda, \mathbb{F}_\mathbb{A}, \mathbb{F}_\mathbb{R}, \delta)$

1. Put all $RED$ states r in $Q$ as $q_r$
2. For all r $\in RED$
   a) For all e $\in$ experimental set
      i. If re $\in RED$ & OT[r][e] = 1
      ii. Then $\mathbb{F}_\mathbb{A} = \mathbb{F}_\mathbb{A} \cup q_{re}$
      i. If re $\in RED$ & OT[r][e] = 0
      ii. Then $\mathbb{F}_\mathbb{R} = \mathbb{F}_\mathbb{R} \cup q_{re}$
   b) For all $q_s \in Q$
      i. For all $q_e \in Q$
         A. For all a $\in \Sigma$
         B. If OT[$q_s$a][e] = OT[$q_e$][e], for all e $\in$ experimental set
         C. Then add transition from $q_s$ to $q_e$ via a to $\delta$

Figure 13: Gold Build Automaton

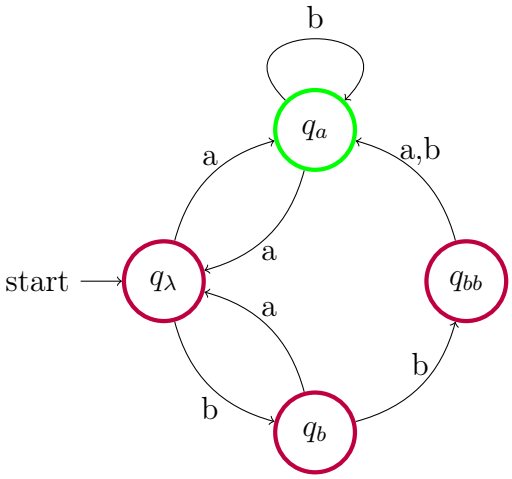


Figure 14: DFA

### 4.3.5. Further Optimisations of Gold

As mentioned before, there are many opportunities to improve Gold's algorithm. One of them is explained here. As any method for improving the result of Gold's algorithm this improvement takes place during the method 'fill holes'.

The shown improvement is an opportunity to collect more informations from the $BLUE$ states. At the first step of 'fill holes' (Figure 8). For each $RED$ state the information contained in the first $BLUE$ state, which is compatible to the $RED$ state, is included into the $RED$ state. During this step the $RED$ state changes. After the inclusion of the additional information contained in $BLUE$, the $RED$ state might not be compatible to other $BLUE$ states, it was compatible before the inclusion. So the other $BLUE$ states don't even get the chance to be a better partner, since this decision is made 'greedily'. So the improvement is, to not just insert the additional of the first compatible $BLUE$ state but take all $BLUE$ states into account and then choose the one, who provides the most additional information. That means, to chose the $BLUE$ state who has the most states defined by a '0' or a '1' where there is a '$*$' at the same experiment at the $RED$ state. This can improve the numbers.

For the example given in this thesis that doesn't matters, because every state contained in $BLUE$ has only one state, which is not equal to '$*$'. So waiting for the $BLUE$ state with most entries in the observation table unequal to '$*$' leads to the case, that all $BLUE$ states which are compatible to the $RED$ state have the same number of entries who delivers new information to the $RED$ state. This can be seen in [Hig10] in the chapter about Gold's algorithm. The example shown there, is following this optimisation but the pseudo-code given there, follows the

same strategy like the implementation in this thesis which is a bit confusing.

## 4.4. RPNI

RPNI (Regular Positive and Negative Inference) is, similar to the Gold algorithm, an algorithm which belongs to the class of informed learner. But in opposition to the Gold algorithm, where there is more than a realistic chance that the result will only be the PTA, and the generalisation fails, RPNI will respond a generalisation in all circumstances[Hig10]. RPNI runs with the following operations:

1. Build the PTA only with the positive set of the input sample (subsubsection 4.4.1)
2. Merge two states, a $BLUE$ one and a $RED$ one (subsubsection 4.4.2)
3. Fold the two states (subsubsection 4.4.3)
4. Check if the new automaton is still compatible to the input (subsubsection 4.4.4)
5. Promote the state, if the automaton is not compatible (subsubsection 4.4.5)

RPNI is a recursive algorithm so it is not that easy like just following the 5 steps, but these are the main operations and explained each in detail in the referenced chapters. At the end in subsubsection 4.4.6, a whole run of the algorithm is shown.

Therefore and also for the examples during explaining every step of the algorithm in detail, the following sample is taken as the input sample:

$$S_+ = \{a, abb, bba, bbb\}$$
$$S_- = \{b, aa, bb, baba\}$$

which is the same sample as taken for Gold algorithm. This is done on purpose, to show the difference in results even for such a small example.

### 4.4.1. Prefix Tree Acceptor

The prefix tree acceptor is build, like it has been explained in subsubsection 4.3.1, just that the set $S_- = \{\}$ (Figure 15). Figure 16 shows the PTA with the initial colouring. Still true for the colouring is, that the border colour of a state indicates if it is an accepting state, than the border is green, or if it is a rejecting state if the border is red. Since all the steps of the algorithm are directly shown on the graph, the states are also coloured, depending on which set they belong to. If the

whole circle of the state is filled red, the state belongs to the set of $RED$ states and if the circle is filled blue, the state belongs to the set of $BLUE$ states. If the circle of the state isn't filled at all, the state doesn't belong to one of the two sets and hasn't been considered until that point of the algorithm. Figure 16 shows the initial coloured PTA.

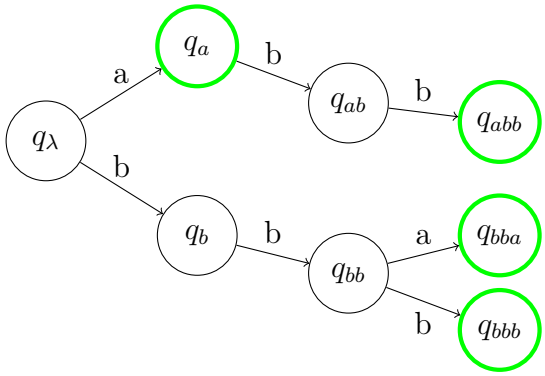


Figure 15: Prefix Tree Acceptor RPNI

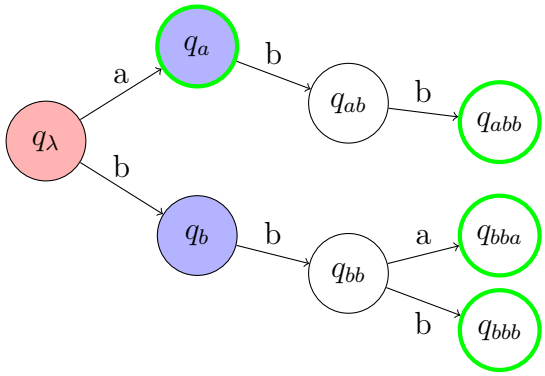


Figure 16: Prefix Tree Acceptor RPNI, initial colouring

### 4.4.2. RPNI Merge

The procedure merge (Figure 17) needs a state $q_r$, which belongs to the set of $RED$ states and a state $q_b$, which belongs to the set of $BLUE$ states. Then merge takes all incoming transition, which is $\{a\}$ in this case, and set the destination of this transition to be $q_r$. It is just owning the fact, that this is the first iteration of the algorithm so $q_r$ is the father node of $q_b$ due to the PTA. That leads to a self loop in $q_r$ with the transition $\{a\}$ Figure 18.

Input: state $r \in RED$, $b \in BLUE$, DFA $A$
Output: DFA $A$ updated

1. for all incoming transitions t of $b$
   a) set t.destination = $r$

Figure 17: RPNI merge

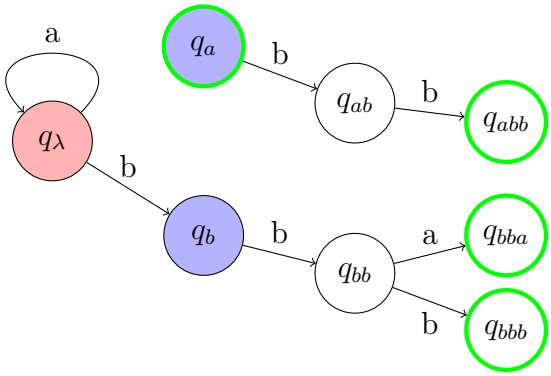


Figure 18: merge of $q_a$ with $q_\lambda$

### 4.4.3. RPNI Fold

RPNI fold folds the sub graph of the $BLUE$ state $q_a$, into the sub graph of the $RED$ state $q_\lambda$. Explained more visual, $q_a$ is placed over $q_\lambda$. So if $q_a$ has been an accepting state, $q_\lambda$ is now an accepting state, since it takes all the properties of $q_a$ and integrates them into himself. Afterwards, every outgoing transition of $q_a$ is tested, if $q_\lambda$ has already an outgoing transition with the same transition value. If that is the case, the algorithm fold is called again, but this time with the two states at the end of the two transitions which is in this case with the two states $q_{ab}$ and $q_b$. This is continued for every outgoing transition in every new sub tree of $q_a$ until $q_a$ is completely fold into $q_\lambda$.

It is shown for the transition b from $q_a$ to $q_{ab}$ in Figure 19 and the result of the fold in Figure 20.

At every transition there is also the possibility to end the recursively called procedure. If the value of the outgoing transition $c$ of $q_a$ is not contained in the outgoing transitions of $q_\lambda$, $c$ is added to the outgoing transitions of $q_\lambda$. Doing this, the hole sub tree of the destination of $\beta$ is connected via $c$ to $q_\lambda$ like it is shown in Figure 21.

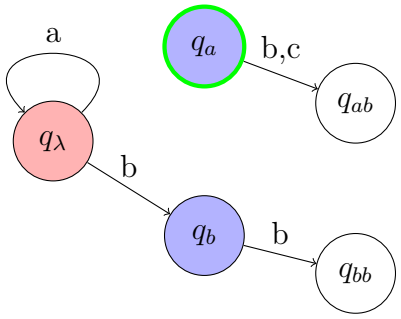


Figure 19: Fold of $q_a$ with $q_\lambda$ exemplary

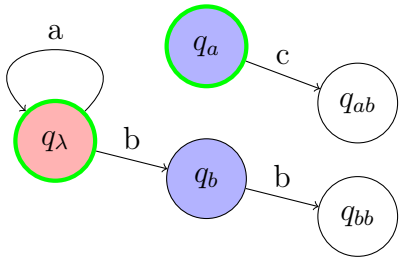


Figure 20: After fold transition b of $q_a$ with $q_\lambda$ exemplary

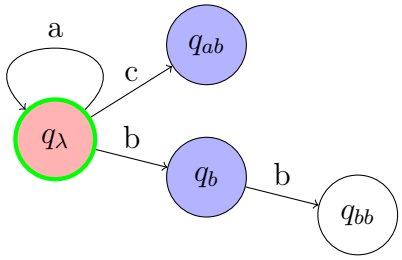


Figure 21: After fold $q_a$ with $q_\lambda$ exemplary

So this step contains the generalisation from a specific input set to an infinite language.

Input: DFA $A$, $RED$ state $q_r$, $BLUE$ state $q_b$
Output: DFA $A$ updated

1. If $q_b \in \mathbb{F}_{\mathbb{A}}$
2. Then $\mathbb{F}_{\mathbb{A}} = \mathbb{F}_{\mathbb{A}} \cup q_r$
3. For all outgoing transitions of $q_b$ (value of transition = $\alpha$)
   a) If $q_r$ has an outgoing transition with value $\alpha$
   b) Then: fold($A$, destination of transition with value $\alpha$ with starting point $q_r$, destination of transition with value $\alpha$ with starting point $q_b$)
   c) Else: add new transition $\alpha$ to $q_r$ with same destination like transition $\alpha$ had before

Figure 22: RPNI fold

### 4.4.4. RPNI Compatible

The result of the first fold operation in the real PTA is shown in Figure 21.The operation 'compatible' is testing, if the result of this fold is still corresponding to the input sample. As a short reminder, the input sample was:

$$S_+ = \{a, abb, bba, bbb\}$$
$$S_- = \{b, aa, bb, bab\}$$

For testing the causality the algorithm 'compatible' takes every entry of $S_-$ and tests if it still doesn't lead to an accepting state in the new automaton. This condition fails for the example bb, which now leads to an accepting state. This means, that the whole fold of the automaton is not corresponding to the input sample and therefore not valid.

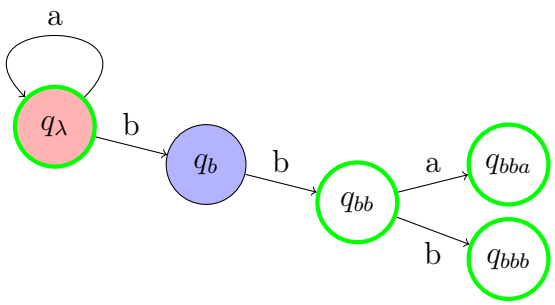


Figure 23: fold of $q_a$ with $q_\lambda$

### 4.4.5. RPNI Promote

If the fold was not valid, and the $BLUE$ state $q_b$ which was tested has been tried to merge with every state contained in $RED$ the tested $BLUE$ state will be promoted to $RED$. Therefore, $q_b$ is deleted in the set of $BLUE$ and added to the set of $RED$ states. Additionally all childes of $q_b$ in the automaton, which doesn't already belong to a coloured set, are added to the $BLUE$ set. The resulting automaton is shown in Figure 24.

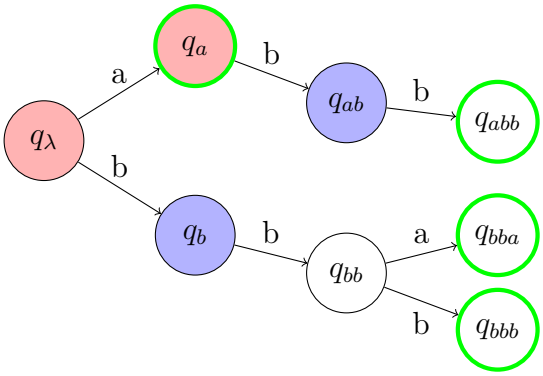


Figure 24: Promotion of $q_a$ to $RED$

This promotion is necessary because there is at least one information contained in $q_b$ that is not already contained in any of the states in $RED$. So if the Information in $q_b$ is deleted, the automaton isn't corresponding to the input sample any more and therefore not describing the same language, because it is accepting a state, the target language wouldn't. So to not loose the information, $q_b$ must be promoted.

### 4.4.6. Example run

In this section, the complete run of RPNI for the given example is shown. All decisions during the whole algorithm will be explained. The structured pseudo code for the whole algorithm is shown in Figure 26.

The first step is to build the prefix tree out of the positive examples as you can see in Figure 15. After that, the initial sets of $RED$ and $BLUE$ are build like it is in Figure 16.

Now it is tried to generalize the PTA. Therefore the first state $q_a \in BLUE$ is chosen for trying to compress the PTA. You can see the merge of $q_a$ with $q_\lambda$ in Figure 18 and the result of the fold in Figure 23. This resulting automaton is not compatible to the input sample because parsing $bb \in S_-$ is leading to an accepting state. Due to the fact that $q_a$ has been not successfully tried to merge with all states contained in $RED$ it is promoted to become a $RED$ state himself as it is shown in Figure 24. All childs of $q_a$ will turn into $BLUE$ states.

The set of $BLUEis\{q_b, q_{ab}\}$, so the next state which has to be chosen is $q_b$ because it has a shorter distance to the root in the original PTA. The merge of the $BLUE$ state $q_b$ with the $RED$ state $q_\lambda$ is shown in Figure 25. The fold of $q_b$ with $q_\lambda$ is displayed in Figure 27 which is also not compatible to the input sample because b in now leading to an accepting state. Now $q_b$ is not promoted to the set of $RED$ states because there is still another state in $RED$ $q_a$, which hasn't been tested. So the next step is to merge and then fold $q_b$ with $q_a$ which is shown in Figure 28. As this is also not consistent to the input as b parses now to true, $q_b$ must also be promoted to $RED$ (Figure 29).

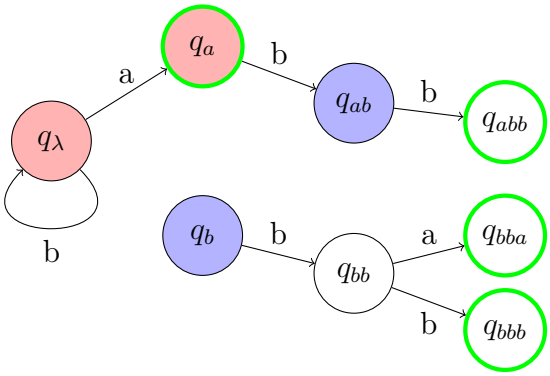


Figure 25: Merge of $q_b$ with $q_\lambda$

Input: $S_+$, $S_-$
Output: DFA $A$

1. $A \leftarrow$ Build PTA (Figure 2)
2. $RED = RED \cup q_\lambda$
3. $BLUE = BLUE \cup$ childes of $q_\lambda$ due to $A$
4. While $BLUE \neq \emptyset$ Do
   a) select next $q_b \in BLUE$ to merge
   b) For all $q_r \in RED$ DO
   c) $B =$ a real copy of $A$
      i. $B =$ merge $(q_r, q_b, B)$ (Figure 17)
      ii. $B =$ fold $(B, q_r, q_b)$ (Figure 22)
      iii. IF $B$ is compatible to $S_-$ (subsubsection 4.4.4)
      iv. Then
         A. $A = B$
         B. $BLUE = Blue$.deleteAll
         C. $BLUE =$ all direct childes of all $q_r \in RED$ that are not $\in RED$
      v. Else:
      vi. Do **not** update $A$, all changes in $B$ are invalid
      vii. If $q_b$ was already tried to merge with all $q_r$ $inRED$
      viii. Then promote $q_b$ (subsubsection 4.4.5)
5. For $q_r \in RED$
   a) For $s \in S_-$
      i. parse $s$ by the automaton
      ii. mark final state of $s$ as rejected

Figure 26: RPNI

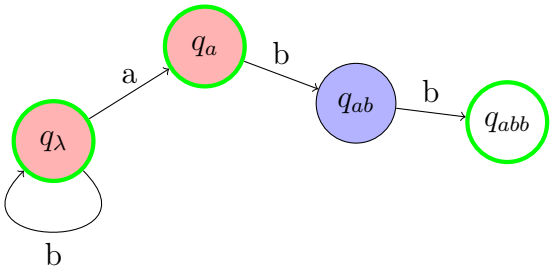


Figure 27: Fold of $q_b$ with $q_\lambda$

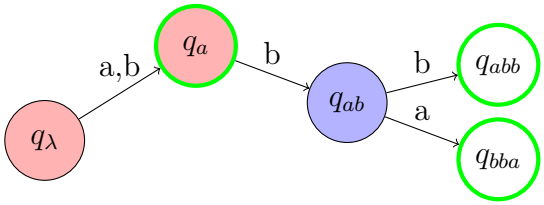


Figure 28: Fold of $q_b$ with $q_a$

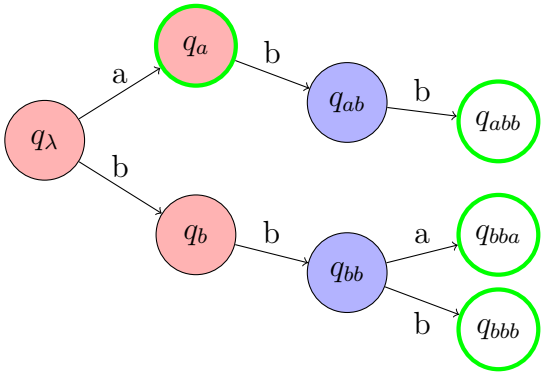


Figure 29: Promotion of $q_b$ to $RED$

The next $BLUE$ state to merge is $q_{ab}$. Merging $q_{ab}$ with $q_\lambda$ isn't successful but merging $q_{ab}$ with $q_a$ is consistently to the input sample. The resulting automaton of folding $q_{ab}$ with $q_a$ is shown in Figure 30. $q_{bb}$ is not merging properly to any of the states contained in $RED$, so it is promoted as shown in Figure 31.

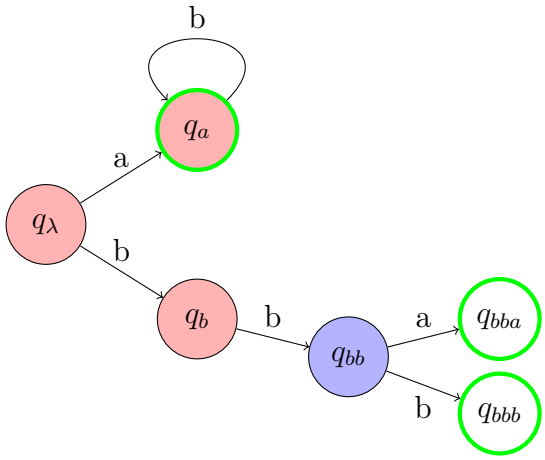


Figure 30: Merge of $q_{ab}$ with $q_a$

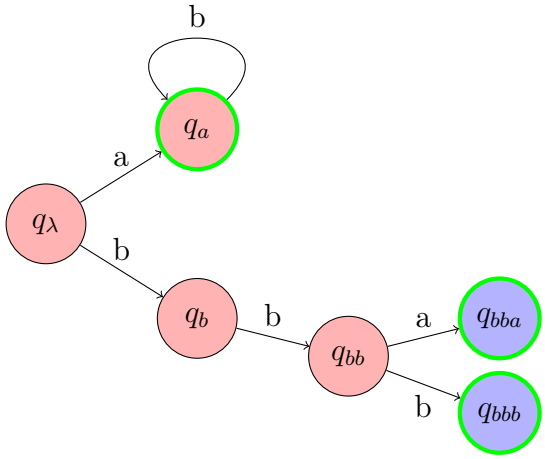


Figure 31: Promote $q_{aa}$ to $RED$

At least the remaining two states $q_{bba}$ and $q_{bbb}$ must be considered. At first $q_bba$ is fold with $q_\lambda$, which is successful (Figure 32). Afterwards $q_{bbb}$ is merged with $q_\lambda$ which is also successful (Figure 33).

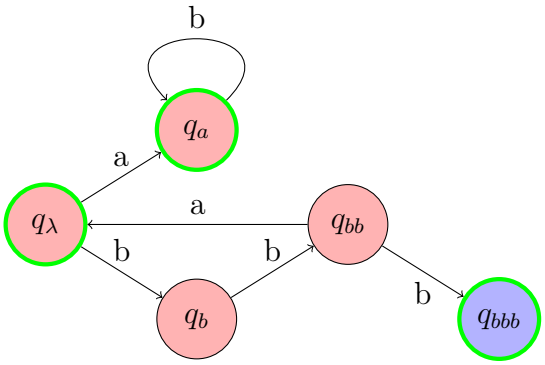


Figure 32: Fold $q_{bba}$ with $q_\lambda$

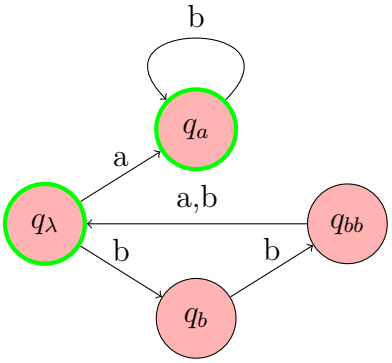


Figure 33: Fold $q_{bbb}$ with $q_\lambda$

The final states and the transitions have been found. The only missing step is to mark the rejecting states. This is done by parsing each negative input sample with the automaton. If there are transitions missing in the automaton to parse the example completely, that is no problem, because the only requirement was, that the automaton should not parse the example and end at an accepting state. If the example parses completely, the final state must be a rejecting state. If the automaton would parse a negative input sample to an accepting state, there must be a mistake in between so at least one fold operation was accepted although it was not allowed. If there was not happening a mistake during the building of the *RED* set and finding the transitions, a negative input can't be parsed to an accepting state because of the check that is done after each fold and the rollback of all changes if the check fails.

So parsing the negative input set leads to the final automaton with the correct accepting and rejecting states. This final automaton is shown in Figure 34.

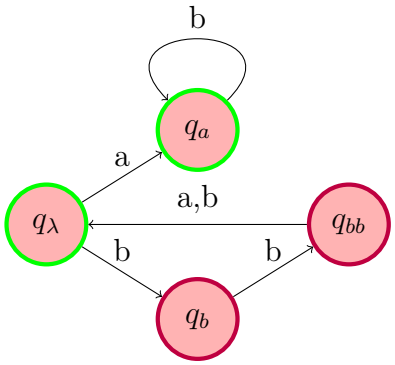


Figure 34: RPNI final DFA

### 4.4.7. Further Optimisation of RPNI

There is also different opportunities to improve the RPNI algorithm. In 2003 Sebban and Janodet published a modification of RPNI, which they named RPNI*. In this modification of RPNI, they tried to relax the rules, for an accepting merge. But additionally not to increase the errors caused by a to optimistic generalisation. So they modified the merging rules, by adding statistical inference theory to the choice, if a merge is accepted. Due to that statistical theory the performances increases significant but the resulting automaton can be wrong because of the statistical choice of accepting merges. [SJ03]

A french researchers group around Jean-Christophe Janodet worked at a new weighting schema for RPNI on the base of confidence oracles. By the help of there real-valued confidence oracle they were able to increase the efficiency of RPNI dramatically. Therefore they introduced a new theoretical boosting scheme which is quite similar to the use of oracles. They prooved, that there weighting scheme has the theoretical properties of boosting. On Base of that they build an oracle which decides if a 'fold' operation is accepted or not. [JNSS04]

# 5. Benchmarks and Result

For running the algorithm on real examples, HÄFNER Gewichte GMBH [1] which is a German weight manufacturer company, provides an example data set of weight numbers. This weight number is always a string which consists of ten characters. The second character is always a dot and the seventh character is a minus. In this weight number different characteristic of the weight are decoded in. For example the first value provides informations about the class, the weight is calibrated for. In weight calibration labours the class 'E0' is the highest accuracy class, where the comparators and the weights are only having very small uncertainty's. Of cause there are a lot more accuracy classes. The whole definition of the weighting classes can be found in [OIM] which is the Organisation International De Métrologie Légale and the most common definition of weights in Europe. There are also other standardisations like ASTM (American Society of Testing & Measurement)[AST] but the company is a German manufacturer so he normal uses the OIML specification.

The second value provides information about the design. This can be for example knob weights or even wire weights. The design is very important since it delivers information about the cross-section-dimension and the hight, which is necessary to know for robot weight comparators.

The raw material is decoded in the third value. In the highest weighting classes the denseness is important to know for calculating the uncertainty.

The other values are decoding other weight specific values which I'm not allowed to publish.

## 5.1. Benchmark

For benchmarking the two algorithms, there are four sets of examples. The size of the example always says that there are this number of positive examples and this number of negative examples in the example set. The four sets are of size 20, 100, 1000 and 5000. Each example is chose randomly from the Database. It is not possible to get a bigger dataset because right now, that was all they gave away. But the algorithms are able to parse bigger datasets it was just nothing bigger available.

### 5.1.1. Benchmark Gold

Gold was running with all four datasets but with no dataset it was possible to calculate a feasible solution except the prefix tree acceptor consisting of the positive input sample. Even finding the small example to explain the algorithm in this

[1] https://www.haefner.de/

thesis needed about 10 tries to find an input sample where the Gold algorithm was able to find a generalisation.

The problem is, when taking a closer look to the observation table, that there are very view entries which are not '$*$'. Since the negative examples of the example set had also always a length of 10 characters there was only very view entries for each state except $q_\lambda$ that was not '$*$'. The observation table was so undermanned, that there was no promotion of a $BLUE$ state to $RED$. So the result had always been $q_\lambda$ with a self loop and the transition had all values of the alphabet $\Sigma$ at transition values. But that leads to a graph, which will accept every input, which consists of the characters of the alphabet. So off cause it also accepts the rejecting input samples and because of that, the resulting automaton is always not accepted and the prefix tree is returned.

So the idea was to modify the input set as long as Gold will be able to deliver a solution. Since all rejecting strings where of length 10, the idea was, to build more rejecting states of shorter length and maybe get a closer insight to how many rejecting states are necessary for a input set of positive examples of a specific sice, to get a feasible generalisation. Since now, there are only entries in the observation table different from '$*$', if the concatenation of the state and the experiment are of size 10 and the result is part of the input sample, it might generate additional informations and therefore entries to add shorter examples to the negative input samples.

Therefore i tried the positive input sample with 100 entries, and added negative examples to the rejecting input sample. After about 40 tries and at least a negative input sample with about 14.000 entries, the algorithm was still not able to calculate a feasible generalisation.

So except very small constructed example sets, the algorithm delivered not one generalisation.

### 5.1.2. Benchmark RPNI

Benchmarking RPNI with the four example sets leads to the following connection between time and the size of the input sample Figure 35. Where the time in in ms and the examples are always the complete example set, so positive and negative samples together.

So RPNI isn't running that bad, even in this implementation. The uncertainty was increasing and at for the 5000 example sample at about 30 seconds. This curve looks like a linear relation, what is not that what i expected after reading that the runtime of the algorithm is quite terrible and that it doesn't scale well [Hig10]. Maybe, the example just wasn't big enough to shown the dramatic increase of runtime and its still in a very relaxed part of increasing the runtime of a different function than linear. That is the only thing that seems to make sense, since in this

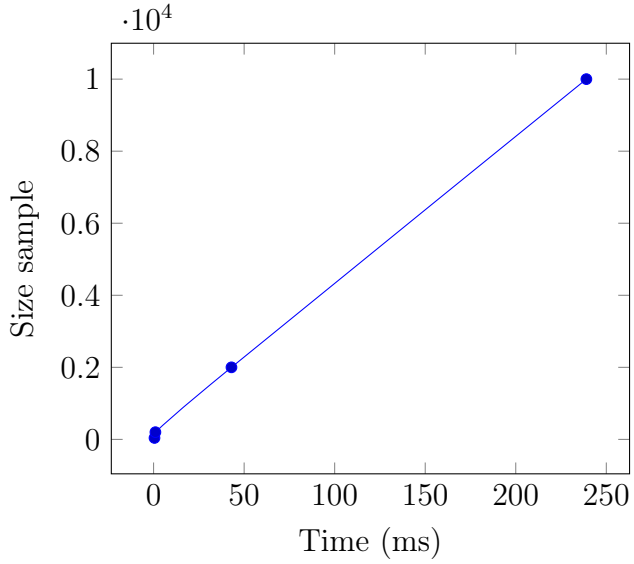


Figure 35: RPNI Benchmark

thesis, no dramatic runtime could be observed.

It would be dramatic if running it at a mobile phone or tablet, but if the server updates the automaton in a real world use case as described and delivers that to its connected devices, it doesn't seem to be unrealistic to use this algorithm. Of cause this is only an example and no proof of the runtime. Expect the statement that the algorithm runs in polynomially time there haven't been other statements about the runtime in literature that was read [Hig10]. Because no bigger or different dataset had been available, there is no information about the general runtime.

## 5.2. Results

The result for Gold algorithm is, that it is not suitable for the problem. It must be questioned, if it is even suitable in the given implementation for learning any big context sensitive language. It is not helpful, that the result of the algorithm is the prefix tree acceptor so often. In context of this thesis, the algorithm wasn't able to deliver one generalisation from a real world dataset. So the only result is, that using this algorithm without a lot of further optimisation of guessing the holes, might not be useful for the given problem.

Running RPNI on input sample consisting only out of positive examples the solution was like shown in Figure 36. That isn't really surprising because the negative examples are those who are needed to deny a fold operation. Since there are no negative examples at all, every fold operation will be valid, and the solution is as shown. There is just the state $q_\lambda$ and a selfeloop from $q_\lambda$ to $q_\lambda$ with every

character from $\Sigma$ being a variable at this transition. So running RPNI only dose make sense, if there are negative examples as well. Otherwise the solution is not helpful at all, to parse a string and check if it is part of the target language or not since the regular expression parses everything if all characters of the parsed string have been part of the input sample. So starting with a ground base of truth with only consisting positive examples is OK, but it should not be delivered to the customer like that. There should be done a training of the algorithm before.

Running RPNI on the input sample of 20 inputs the result was like shown in Figure 37.

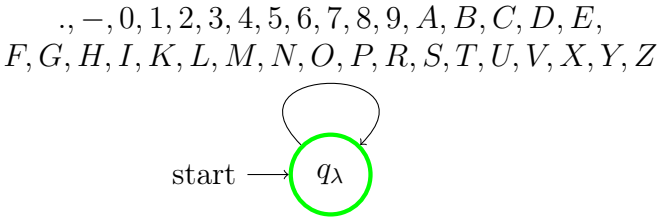


Figure 36: Result RPNI Example set of only positive examples

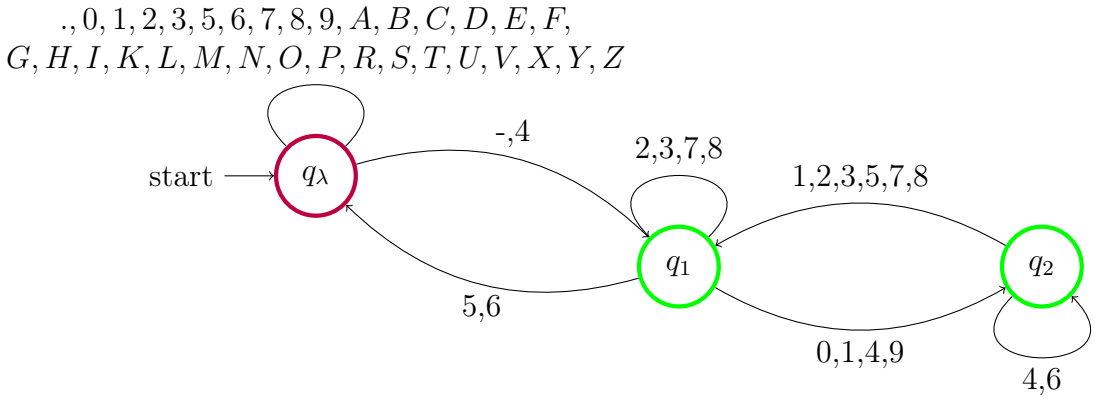


Figure 37: Result RPNI Example set of 20

In Figure 39 the resulting automaton is shown. To increase the readability the notation had been changed for this automaton. The accepting and rejecting states are marked by the filling colour of the state. The accepting states are filled green, the rejecting states are filled violet. The outer circle of a state has the same colour as the outgoing transitions from that state.

The graph doesn't show a lot of generalisation at the alphabet part of the input. This might occur to the input sample, and that the first three characters in the

input string, can have nearly every letter or digit which is possible. This leads to the big self loop of $q_\lambda$. But you can see, that no input sample can be parsed just by $q_\lambda$ because the '-' is not part of the transition values at the self loop. So '-' can be seen as a point in each sample, that separates the graph a little bit even if it can be reached in different ways. The '.' was not seen as a separator although it is also a member of each sample at the same point. Maybe there just had been far to less input examples, for identifying the '.' as a separator, or maybe that is also a problem of the missing rejecting input samples of different length.

This shows that even with a small input size, the graph will get big.

The result of the input sample of size 1000 is not drawable because it has 65 states. So the level of detail is increasing immense. Therefore especially the credits go to the size of the negative input sample. Since this set of examples is avoiding folds in the graph, the more negative states there are in the set, the higher is the probability that one state prohibits the fold. But it is shown in subsubsection A.1.1 in the Appendix.

There can be seen, that also the first part of the sample is separated more. The alphabet is more divided into different states and not just contained in one self loop. So the automate is getting more precise.

The result of the example set of size 5.000 is surprising since it is near as small as the result of the example consisting of 20 examples. But taking a closer lock on it, it attracts attention, as it is really structured. This graph is separated by the '-' like is was expected to be. Also this is the first time, where not each state is either a member of the accepting states or the rejecting states. So also the length of the input get more attention. After crossing the '-' transition from state $q_\lambda$ to $q_1$ there must be at least three additional characters until the input is accepted again. So this is the first example which really provides information about the length of the accepting strings. Of cause there could be more, like separating also at the '.' but this is only an example of size 5000 and all examples both negative and positive have all the same length. Additionally, the states, which have not been dedicated to either the accepting or rejecting states are still unknown. There are to view examples to decide if they belong to one set or the other.

Probably, adding more examples and more different counter examples also of different length would make the automaton more precise. This doesn't have to be the same like increasing the number of states. The result of the example consisting 5000 inputs is much smaller and at the same time more precise.

So the expectation is, that adding negative examples makes the resulting automaton more precise and adding positive examples, makes the result smaller. Of cause, this is a real world dataset and therefore making predictions is very hard, but the tendency seems to be like that. And that would also make sense because more negative input samples avoid more folds but more positive input samples

provide more possibilities to fold the automaton.

Finally the question of the membership problem from section 2 has still to be answered. If there is an input string 's' than character after character the right transition must be followed. If at some point, 's' still has characters, for which no transition has been taken, but at the actual point, there is no transition with the same transition value of the actual character of 's', than 's' doesn't belong to the language. If 's' is completely parsed by the automaton, and ends at an accepting state, 's' is part of the language. If 's' is completely parsed and ends at a rejecting state, 's' is for sure not part of the language. If the ending state is still neutral, there are information missing and the decision is unclear on the base of the given data set. Then the use case comes into account. Is it better to accept a wrong state or reject a positive state. That is a decision which has to be made individual for the given use case since it is not decidable on base of the given data set. Nevertheless, parsing the example is able to do in linear runtime to the respective to the size of the input sample which has to be checked. This is possible because the automaton is deterministic, so each transition value is only appearing at least one time at an outgoing transition from each state.

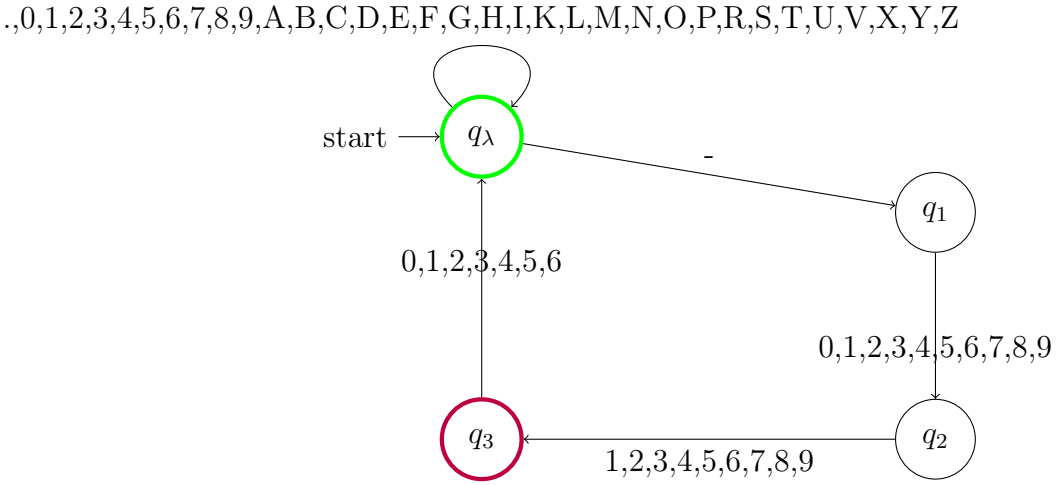


Figure 38: Result RPNI Example set of 5.000

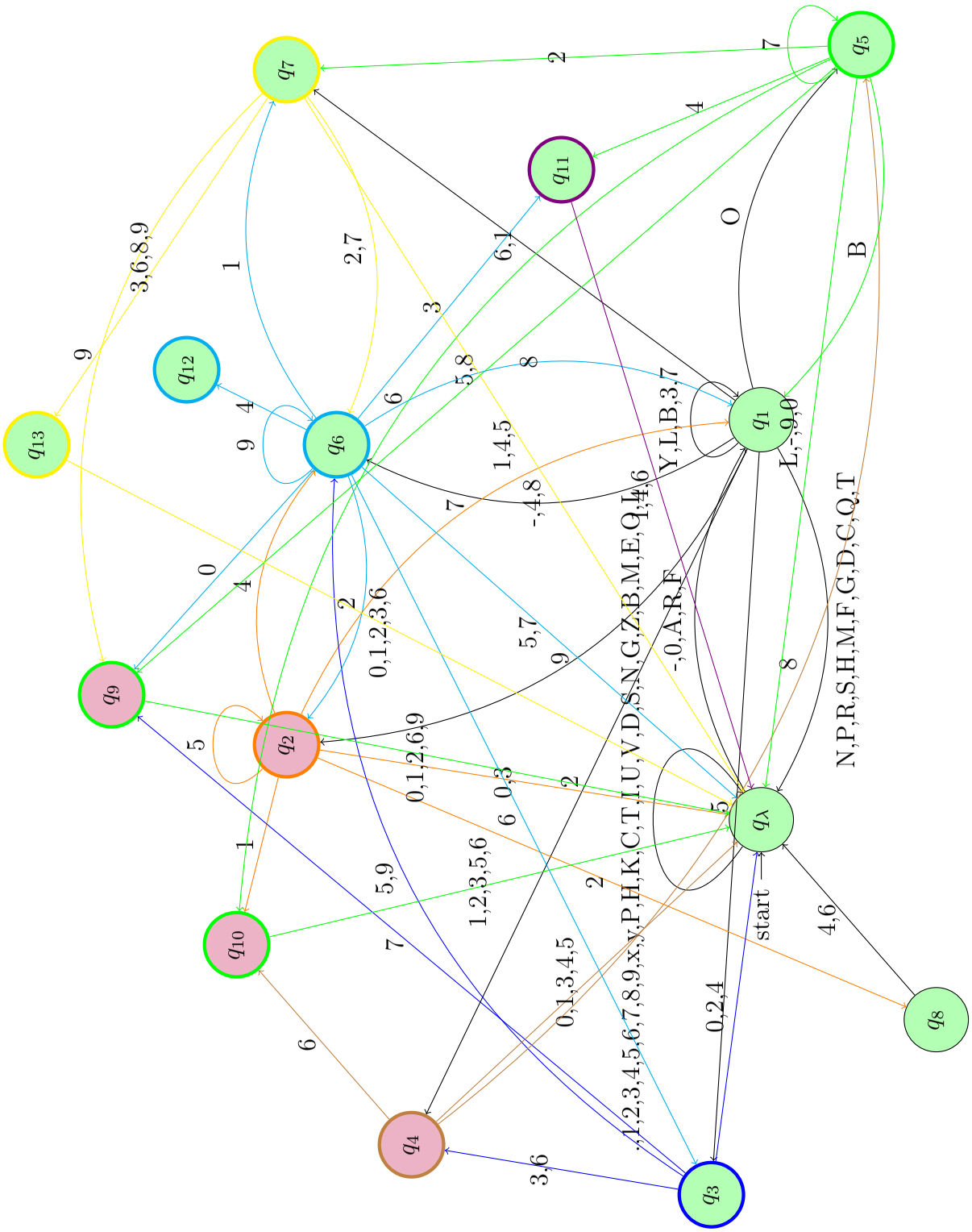


Figure 39: Result RPNI Example set of 100

# 6. Closure

In this thesis different strategies to learn regular expressions had been investigated. After a short introduction to the different methods learning with automatons had been focused on. Due to unpredictable difficulties with the optical character recognition software Tesseract and the learning algorithm DeLeTe2 the usability of the two algorithms Gold and RPNI for learning real world problems had been analysed.

Summing up the results of this thesis, there is to say that the Gold algorithms is not convenient to learn a real world context sensitive input sample consisting of positive and negative examples. For each tested example set, the Gold algorithm wasn't able to made any generalisation at all. Because getting a generalisation at all at not just the prefix tree acceptor of the positive input samples is so seldom, the algorithm isn't a good choice for learning real world data sets where the input is not very predictable.

RPNI delivered every time a generalisation. But this one wasn't always comprehensible due to the size of the resulting automaton. Id was interesting, that after the automaton increases with the size of the input sample, at a size of 5000 input samples, the automaton decreases dramatically from 65 states with 1000 input samples to 4 with 5000 input samples. This leads to the conclusion, that increasing the size of the negative input samples increases the resulting automaton. Increasing the positive input sample, the automaton decreases with a little delay due to the fact of the quantity of fol operations that are tested and to the number of fold operations that result to an automaton that is still conform to the input sample.

Benchmarking the RPNI algorithm results to a linear relation between runtime and the size of the input sample. But the examples haven't been that big because nothing else had been available. So maybe this was just a nearly linear part of a different function with an increasing rise. In the experiment made in this thesis the relation was linear but it was only made with one real world example set due to le lack of other opportunities.

For the given use case, RPNI can be helpful as against to Gold's algorithm. RPNI is able to give generalized automaton as a result. But since the calculation of this automaton is a lot of calculating, every tablet should send there newly gain input sample to the server, who updates the automaton from time to time. After updating the automaton, the result can be send back to the tablets. Since the calculation is calculating intense, this shouldn't be done at the clients but at the more powerful server.

# A. Appendix

## A.1. Graph descriptions

Like explained before, the description of the graph has to be read as following:
2.2G : State: ACCEPTED,childs →
-> X
2.2G -> G
2.2G -> C

State 2.2G is an accepting end state. The following list is a list of its direct neigbours which are reachable from 2.2G and the transitions to them. When nothing is standing there, that means state $\lambda$. So $\lambda$ is reachable from 2.2G via 'X'. And there is a self loop at 2.2G with the transition values 'C' and 'G'.

Every transition in this graph has its own row and the definition of the states have also there own row. The numbering of the state is a result from the fold methods. This was the label of the state at the prefix tree acceptor. The result is coming directly from the implementation.

### A.1.1. 1.000 example input sample

: State: ACCEPTED, childes ->
-> X
-> 4
-> 5
-> 1
-> 3
-> 7
-> Y
-> 6
-> 2
-> 8
-> 9
-> .
-> Z
-> I
-> K
-> V
-> D
-> U
-> B

-> N
2.2G -> G
2.2G -> C
2.2G -> 0
2.2G -> E
2.2G -> M
2.2G -> A
2.2G -> H
2.2G -> P
2.2G -> R
2.2G -> S
2.2G -> -
5.3ZL -> L
5.3ZL -> F
5.3ZL -> T
5.3ZL -> O
2.2G : State: ACCEPTED, childes ->
-> A
2.2G -> Z
2.2G -> E
2.2G -> P
2.2G -> G
2.2G -> C
2.2G -> 7
5.YGL -> L
5.YGL -> R
5.YGL -> B
5.YGL -> K
5.YGL -> -
5.YGL -> 4
5.3ZL -> N
5.3ZL -> M
5.3ZL -> S
5.3ZL -> F
5.3ZL -> 0
5.3ZL -> H
5.3ZL -> 3
5.3ZL -> 5
5.7GY -> Y
5.7GY -> O

5.7GY -> D
5.7GY -> 6
5.7GY -> 8
5.7GY -> 2
2.2GT -> T
2.2GT -> 1
2.2GT -> 9
5.YGL : State: REJECTED, childes ->
-> H
-> F
-> G
2.2G -> N
2.2G -> O
5.YGL -> S
5.YGL -> R
5.YGL -> T
5.3ZL -> Y
5.3ZL -> C
5.3ZL -> K
5.3ZL -> 8
5.7GY -> L
2.2GT -> B
2.2GT -> D
2.2GT -> M
2.2GT -> P
2.2GT -> -
4.2NLL -> 9
4.ZBFB -> 0
4.ZBFB -> 5
4.ZBFB -> 7
6.1ZLP -> 1
6.1ZLP -> 3
2.DYLT -> 6
9.7BLC -> 4
2.5ZNM-2 -> 2
5.3ZL : State: REJECTED, childes ->
-> Z
-> 3
-> 7
2.2G -> E

2.2G -> F
5.YGL -> H
5.YGL -> 4
5.3ZL -> N
5.3ZL -> A
5.3ZL -> G
5.7GY -> K
5.7GY -> M
2.2GT -> 0
2.2GT -> O
2.2GT -> R
4.2NLL -> L
4.2NLL -> S
4.2NLL -> D
4.2NLL -> -
4.2NLL -> 5
4.ZBFB -> B
4.ZBFB -> Y
4.ZBFB -> 2
6.1ZLP -> P
6.1ZLP -> 6
2.DYLT -> T
9.7BLC -> C
5.1ZGT- -> 9
1.8APR-5 -> 1
5.7GY : State: ACCEPTED, childes ->
-> Y
-> 2
-> 8
-> 7
2.2G -> F
2.2G -> H
5.YGL -> L
5.YGL -> O
5.3ZL -> N
5.3ZL -> M
5.3ZL -> G
5.3ZL -> D
5.3ZL -> 6
5.3ZL -> 4

5.7GY -> S
5.7GY -> T
5.7GY -> R
5.7GY -> 0
4.2NLL -> C
4.2NLL -> P
4.2NLL -> K
4.ZBFB -> B
6.1ZLP -> 9
9.7BLC -> 1
2.7ZMY- -> -
2.5ZNM-2 -> 3
2.2GT : State: ACCEPTED, childes ->
2.2G -> F
2.2G -> P
2.2G -> Y
2.2G -> K
5.YGL -> O
5.YGL -> B
5.YGL -> M
5.3ZL -> S
5.3ZL -> T
5.3ZL -> D
5.3ZL -> L
5.3ZL -> C
5.7GY -> R
7.B0T0 -> 0
7.B0T0 -> 7
5.1ZGT- -> -
5.1ZGT- -> 3
2.7ZMY- -> 9
4.YGGL-6 -> 6
1.8APR-5 -> 5
2.5APL-4 -> 4
2.5APL-4 -> 1
2.5ZNM-2 -> 8
2.8YYY-12 -> 2
4.2NLL : State: ACCEPTED, childes ->
-> H
-> M

-> T
-> B
2.2G -> S
2.2G -> F
2.2G -> Y
5.3ZL -> R
5.3ZL -> D
5.3ZL -> C
7.B0T0 -> 5
6.1ZLP -> 8
4.2NLL- -> -
5.1ZGT- -> 4
4.4ZYT-2 -> 2
4.7BBT-0 -> 0
4.YGGL-6 -> 3
5.MANL-7 -> 7
Y.BBHM-9 -> 9
2.DNNL-6 -> 6
9.DYNL-1 -> 1
4.ZBFB : State: ACCEPTED, childes ->
-> H
-> O
-> S
2.2G -> L
2.2G -> F
2.2G -> K
2.2G -> B
2.2G -> T
2.2G -> D
2.2G -> 6
5.YGL -> R
5.3ZL -> P
5.3ZL -> 0
5.3ZL -> C
5.3ZL -> 4
5.3ZL -> 2
5.7GY -> M
6.1ZLP -> 9
9.7BLC -> 1
4.ZBFB- -> -

2.7ZMY- -> 5
2.5APL-4 -> 8
2.8YYY-12 -> 7
2.8YYY-12 -> 3
7.B0T0 : State: REJECTED, childes ->
-> 0
5.3ZL -> 7
9.7BLC -> 2
6.1ZLP- -> 9
5.MANL-7 -> 1
9.DYNL-1 -> -
9.DYNL-1 -> 6
9.DYNL-1 -> 5
9.DYNL-1 -> 8
7.UELT-03 -> 3
2.8YYY-12 -> 4
6.1ZLP : State: REJECTED, childes ->
-> N
-> F
-> Y
-> D
-> K
-> 3
-> 1
2.2G -> P
5.YGL -> M
5.YGL -> 7
5.3ZL -> L
4.2NLL -> 9
4.2NLL- -> 5
4.2NLL- -> 4
6.1ZLP- -> -
4.4ZYT-2 -> 8
4.7BBT-0 -> 2
4.7BBT-0 -> 6
2.DYLT : State: ACCEPTED, childes ->
-> L
-> K
-> D
-> T

-> Y
-> 1
-> 9
-> 5
2.2G -> C
5.3ZL -> B
5.3ZL -> S
5.3ZL -> 3
5.MANL-7 -> -
9.7BLC : State: REJECTED, childes ->
-> 6
-> 3
-> 4
-> 8
5.1ZGT- -> 1
2.DNNL-6 -> -
2.DNNL-6 -> 2
4.2NLL- : State: ACCEPTED, childes ->
6.1ZLP- -> 8
4.2NLL-2 -> 2
4.2NLL-2 -> 7
5.MPML-5 -> 5
5.MANL-7 -> 4
5.V0FS-6 -> 6
7.N0GY-6 -> 0
2.7YFL-1 -> 1
8.5SGY-0 -> 9
8.CNNS-3 -> 3
4.ZBFB- : State: REJECTED, childes ->
4.YGGL-6 -> 9
5.MPML-5 -> 7
5.V0FS-6 -> 5
5.1ZGT-0 -> 0
5.1ZF0-9 -> 3
1.UENB-1 -> 1
1.UENB-1 -> 6
Y.3BMY-1 -> 2
2.ZSNB-4 -> 4
7.IGLT-43 -> 8
5.1ZGT- : State: ACCEPTED, childes ->

5.3BPT-7 -> 7
5.1ZGT-0 -> 0
5.1ZF0-9 -> 9
5.1ZF0-9 -> 3
1.4Z0T-1 -> 1
Y.5NRY-9 -> 4
Y.5NRY-9 -> 6
Y.BBHM-9 -> 5
2.7ZMY-3 -> 2
9.IBGT-8 -> 8
6.1ZLP- : State: REJECTED, childes ->
-> 1
5.7GY -> 0
5.1ZGT-0 -> 2
1.BASP-4 -> 4
Y.3BMY-1 -> 3
2.ZSNB-4 -> 7
9.7BLC-0 -> 5
7.IGLT-43 -> 6
7.UELT-03 -> 9
2.7ZMY- : State: REJECTED, childes ->
4.ZBFB- -> 7
6.1ZLP- -> 4
4.NNMD-2 -> 2
4.NNMD-2 -> 8
7.N0GY-6 -> 6
Y.5NRY-9 -> 9
Y.3BMY-1 -> 1
6.2BMY-5 -> 5
2.7ZMY-3 -> 3
8.5SGY-0 -> 0
4.NNMD-2 : State: REJECTED, childes ->
-> 9
-> 1
2.2G -> 2
2.DYLT -> 8
4.2NLL- -> 5
5.V0FS-6 -> 6
1.UENB-1 -> 3
9.7BLC-0 -> 4

4.2NLL-2 : State: ACCEPTED, childes ->
-> 1
-> 8
-> 9
-> 0
5.YGL -> 7
2.2GT -> 2
2.DNNL-6 -> 6
X.8YMS-77 -> 5
4.HGHL-24 -> 4
2.8YYY-12 -> 3
4.4ZYT-2 : State: ACCEPTED, childes ->
-> 1
-> 9
-> 0
2.2G -> 8
6.1ZLP -> 6
9.7BLC -> 5
4.2NLL-2 -> 3
9.IBGT-8 -> 2
4.7BBT-0 : State: REJECTED, childes ->
-> -
-> 6
-> 8
-> 7
-> 5
5.3ZL -> 2
5.3ZL -> 9
9.7BLC -> 3
9.7BLC -> 4
4.7BBT-0 -> 1
4.YGGL-6 : State: REJECTED, childes ->
-> 3
-> 1
5.3ZL -> 5
6.1ZLP -> 9
4.ZBFB- -> 8
5.3BPT-7 -> 6
5.3BPT-7 -> 2
2.5APL-4 -> 4

5.MPML-5 : State: ACCEPTED, childes ->
-> 4
-> 0
2.2G -> 8
5.3ZL -> 5
5.1ZGT- -> 7
9.DYNL-1 -> 6
3.MPFS-53 -> 3
5.MANL-7 : State: ACCEPTED, childes ->
-> 4
4.2NLL -> 8
9.7BLC -> 0
5.1ZGT- -> 3
4.2NLL-2 -> 6
6.2BMY-5 -> 5
2.DNNL-6 -> 9
2.DNNL-6 -> 1
8.5SGY-0 -> 2
X.8YMS-77 -> 7
5.V0FS-6 : State: ACCEPTED, childes ->
-> 6
-> 1
-> 8
2.2G -> 5
2.2G -> 3
5.3ZL -> 2
4.ZBFB -> 7
2.DNNL-6 -> 9
4.HGHL-24 -> 4
5.3BPT-7 : State: REJECTED, childes ->
-> 7
-> 0
6.1ZLP -> 4
9.7BLC -> 3
4.NNMD-2 -> 5
5.MANL-7 -> 6
2.5ZNM-2 -> 1
X.PNGK-56 -> 8
5.1ZGT-0 : State: REJECTED, childes ->
-> 1

-> 6
-> 2
-> 8
5.YGL -> 4
9.7BLC -> 9
1.8APR-5 -> 7
2.DNNL-6 -> 5
4.IGLB-03 -> 3
5.1ZF0-9 : State: ACCEPTED, childes ->
6.1ZLP -> 1
5.V0FS-6 -> 4
5.1ZGT-0 -> 5
1.8APR-5 -> 7
Y.BBHM-9 -> 3
X.PNGK-56 -> 6
Y.4GLM-92 -> 2
2.8YYY-12 -> 8
1.UENB-1 : State: REJECTED, childes ->
-> 1
-> 3
-> 9
-> 0
2.2GT -> 5
4.ZBFB -> 6
4.ZBFB -> 2
9.7BLC -> 8
5.1ZGT- -> 4
1.8APR-5 : State: REJECTED, childes ->
-> 9
-> 1
-> 2
2.2G -> 8
5.3ZL -> 4
9.7BLC -> 3
9.7BLC -> 5
4.ZBFB- -> 7
2.DNNL-6 -> 6
1.4Z0T-1 : State: REJECTED, childes ->
-> 2
-> 3

-> 6
2.2G -> 5
2.2GT -> 9
2.2GT -> 1
4.2NLL- -> 8
9.DYNL-1 -> 4
1.BASP-4 : State: ACCEPTED, childes ->
-> 3
-> 6
-> 0
-> 1
7.N0GY-6 : State: REJECTED, childes ->
-> 4
-> 5
6.1ZLP -> 9
9.7BLC -> 6
5.MANL-7 -> 1
5.MANL-7 -> 7
1.8APR-5 -> 8
4.50NM-63 -> 3
2.8YYY-12 -> 2
Y.5NRY-9 : State: ACCEPTED, childes ->
-> 3
-> 7
-> 2
6.1ZLP -> 4
4.2NLL- -> 1
6.1ZLP- -> 8
1.BASP-4 -> 6
Y.5NRY-9 -> 9
2.8YYY-12 -> 5
Y.3BMY-1 : State: ACCEPTED, childes ->
4.2NLL- -> 5
2.7ZMY- -> 3
4.2NLL-2 -> 8
4.4ZYT-2 -> 7
1.UENB-1 -> 2
2.DNNL-6 -> 9
4.50NM-63 -> 1
7.UELT-03 -> 4

Y.BBHM-9 : State: ACCEPTED, childes ->
-> 3
-> 6
-> 0
5.3ZL -> 8
5.3ZL -> 1
5.3ZL -> 5
2.DYLT -> 9
5.1ZGT- -> 2
5.MANL-7 -> 7
2.DNNL-6 -> 4
6.2BMY-5 : State: REJECTED, childes ->
-> 3
-> 8
-> 2
-> 1
2.2GT -> 7
6.1ZLP -> 5
6.1ZLP -> 9
2.DYLT -> 4
X.PNGK-56 -> 6
2.5APL-4 : State: ACCEPTED, childes ->
-> 9
-> 7
-> 0
9.7BLC -> 1
4.ZBFB- -> 6
4.4ZYT-2 -> 8
2.DNNL-6 -> 4
9.DYNL-1 -> 2
7.IGLT-43 -> 3
2.ZPGK-45 -> 5
2.5ZNM-2 : State: REJECTED, childes ->
-> 8
-> 0
5.3ZL -> 5
4.ZBFB -> 7
9.7BLC -> 4
4.ZBFB- -> 2
1.8APR-5 -> 9

Y.BBHM-9 -> 3
2.8YYY-12 -> 1
2.ZSNB-4 : State: ACCEPTED, childes ->
-> 5
-> 6
2.2G -> 8
5.YGL -> 1
2.2GT -> 3
9.7BLC -> 4
2.7ZMY-3 : State: REJECTED, childes ->
-> 7
-> 1
-> 4
-> 5
-> 0
2.2G -> 9
1.UENB-1 -> 2
2.DNNL-6 -> 8
2.7ZMY-33 -> 3
2.7YFL-1 : State: REJECTED, childes ->
-> 5
-> 8
-> 4
5.7GY -> 6
2.2GT -> 7
9.7BLC -> 3
Y.BBHM-9 -> 9
6.2BMY-5 -> 1
2.DNNL-6 : State: REJECTED, childes ->
4.2NLL -> 7
2.DYLT -> 5
4.2NLL- -> 2
2.7ZMY- -> 3
4.4ZYT-2 -> 9
5.MPML-5 -> 1
5.1ZF0-9 -> 4
Y.3BMY-1 -> 6
Y.BBHM-9 -> 8
9.7BLC-0 -> 0
8.5SGY-0 : State: ACCEPTED, childes ->

-> 9
-> 0
6.1ZLP -> 6
9.7BLC -> 3
5.1ZGT- -> 7
6.1ZLP- -> 1
1.4Z0T-1 -> 4
2.5ZNM-2 -> 5
2.7YFL-1 -> 8
9.DYNL-1 -> 2
8.CNNS-3 : State: REJECTED, childes ->
2.DYLT -> 3
9.DYNL-1 : State: REJECTED, childes ->
-> 7
2.7ZMY- -> 9
4.NNMD-2 -> 4
4.7BBT-0 -> 8
Y.BBHM-9 -> 0
2.5APL-4 -> 3
2.5APL-4 -> 1
9.IBGT-8 -> 2
X.8YMS-77 -> 5
2.ZPGK-45 -> 6
9.7BLC-0 : State: ACCEPTED, childes ->
-> 1
-> 0
2.2G -> 9
4.2NLL -> 5
4.ZBFB -> 2
5.1ZGT- -> 6
7.1EMC-04 -> 4
9.IBGT-8 : State: ACCEPTED, childes ->
5.7GY -> 1
2.2GT -> 2
5.1ZGT- -> 4
X.8YMS-77 : State: ACCEPTED, childes ->
-> 7
-> 5
-> 0
-> 6

-> 2
-> 3
2.7ZMY- -> 4
X.PNGK-56 : State: REJECTED, childes ->
-> 0
-> 3
-> 4
-> 6
4.50NM-63 : State: ACCEPTED, childes ->
-> 2
-> 5
-> 4
-> 3
-> 1
-> 0
4.IGLB-03 : State: ACCEPTED, childes ->
-> 0
-> 2
-> 4
4.HGHL-24 : State: ACCEPTED, childes ->
-> 0
-> 4
-> 5
-> 6
-> 2
3.MPFS-53 : State: REJECTED, childes ->
-> 4
-> 5
-> 6
-> 2
7.IGLT-43 : State: ACCEPTED, childes ->
-> 6
-> 3
-> 1
-> 4
-> 0
9.7BLC -> 2
7.1EMC-04 : State: REJECTED, childes ->
-> 1
-> 3

7.UELT-03 : State: REJECTED, childes ->
-> 1
-> 6
-> 0
-> 4
-> 5
1.8APR-5 -> 3
Y.4GLM-92 : State: ACCEPTED, childes ->
-> 3
-> 1
-> 2
2.ZPGK-45 : State: ACCEPTED, childes ->
-> 0
-> 5
-> 4
-> 1
2.8YYY-12 : State: REJECTED, childes ->
-> 1
-> 2
-> 0
-> 6
2.2G -> 8
5.YGL -> 3
2.5ZNM-2 -> 9
2.DNNL-6 -> 5
2.7ZMY-33 : State: ACCEPTED, childes ->
-> 5
-> 0
-> 4
-> 6

# B. List of Figures


1. Prefix Tree Acceptor Gold . . . 10
2. Build prefix tree acceptor . . . 11
3. Observation table $RED = \{\lambda\}$ . . . 12
4. Fill in observation table . . . 13
5. Observation table $RED = \{\lambda, a\}$ . . . 13
6. Observation table $RED = \{\lambda, a, b\}$ . . . 14
7. Observation table $RED = \{\lambda, a, b, bb\}$ . . . 15
8. Fill holes . . . 15
9. Observation table after first step of fill holes . . . 16
10. Observation table after second step of fill holes . . . 17
11. Observation table after last step of fill holes . . . 17
12. Transition matrix . . . 19
13. Gold Build Automaton . . . 19
14. DFA . . . 20
15. Prefix Tree Acceptor RPNI . . . 22
16. Prefix Tree Acceptor RPNI, initial colouring . . . 22
17. RPNI merge . . . 23
18. merge of $q_a$ with $q_\lambda$ . . . 23
19. Fold of $q_a$ with $q_\lambda$ exemplary . . . 24
20. After fold transition b of $q_a$ with $q_\lambda$ exemplary . . . 24
21. After fold $q_a$ with $q_\lambda$ exemplary . . . 24
22. RPNI fold . . . 25
23. fold of $q_a$ with $q_\lambda$ . . . 26
24. Promotion of $q_a$ to $RED$ . . . 26
25. Merge of $q_b$ with $q_\lambda$ . . . 27
26. RPNI . . . 28
27. Fold of $q_b$ with $q_\lambda$ . . . 29
28. Fold of $q_b$ with $q_a$ . . . 29
29. Promotion of $q_b$ to $RED$ . . . 29
30. Merge of $q_{ab}$ with $q_a$ . . . 30
31. Promote $q_{aa}$ to $RED$ . . . 30
32. Fold $q_{bba}$ with $q_\lambda$ . . . 31
33. Fold $q_{bbb}$ with $q_\lambda$ . . . 31
34. RPNI final DFA . . . 32
35. RPNI Benchmark . . . 35
36. Result RPNI Example set of only positive examples . . . 36
37. Result RPNI Example set of 20 . . . 36
38. Result RPNI Example set of 5.000 . . . 38
39. Result RPNI Example set of 100 . . . 39

# C. References

[AST] *ASTM.* http://www.uniquetek.com/store/696296/uploaded/calibration_weight_tolerances.pdf, Abruf: Dezember 2018

[bug] *GitHub.* https://github.com/tesseract-ocr/tesseract/issues/403, Abruf: November 2018

[Cho56] Chomsky, Noam: Three models for the description of language. In: *IRE Transactions on information theory* 2 (1956), Nr. 3, S. 113–124

[DLT04] Denis, François ; Lemay, Aurélien ; Terlutte, Alain: Learning regular languages using RFSAs. In: *Theoretical computer science* 313 (2004), Nr. 2, S. 267–294

[Geo] *reddit.* https://www.reddit.com/r/MachineLearning/comments/2lmo0l/ama_geoffrey_hinton., Abruf: Dezember 2018

[GMC+92] Giles, C L. ; Miller, Clifford B. ; Chen, Dong ; Chen, Hsing-Hen ; Sun, Guo-Zheng ; Lee, Yee-Chun: Learning and extracting finite state automata with second-order recurrent neural networks. In: *Neural Computation* 4 (1992), Nr. 3, S. 393–405

[Gol] *Gold's Algorithm.* http://pagesperso.lina.univ-nantes.fr/~cdlh/slides/allpdf/Gold_s_algorithm.pdf, Abruf: November 2018

[GRC] Garcia, Pedro ; Ruiz, José ; Cano, Antonio: Non Deterministic RPNI.

[Hig10] Higuera, Colin De l.: *Grammatical inference: learning automata and grammars.* Cambridge University Press, 2010

[JNSS04] Janodet, Jean-Christophe ; Nock, Richard ; Sebban, Marc ; Suchier, Henri-Maxime: Boosting grammatical inference with confidence oracles. In: *Proceedings of the twenty-first international conference on Machine learning* ACM, 2004, S. 54

[Kin08] Kinber, Efim: On learning regular expressions and patterns via membership and correction queries. In: *International Colloquium on Grammatical Inference* Springer, 2008, S. 125–138

[lea] *Lern Psychologie.* http://www.lern-psychologie.de/common/lernen.htm, Abruf: Dezember 2018

[Man15] Manning, Christopher D.: Computational linguistics and deep learning. In: *Computational Linguistics* 41 (2015), Nr. 4, S. 701–707

[MAR] *reddit*. https://www.maro.de/index.html, Abruf: Dezember 2018

[OIM] *OIML*. https://www.oiml.org/en/files/pdf_r/r111-1-e04.pdf, Abruf: Dezember 2018

[Sak92] Sakakibara, Yasubumi: Efficient learning of context-free grammars from positive structural examples. In: *Information and Computation* 97 (1992), Nr. 1, S. 23–60

[SJ03] Sebban, Marc ; Janodet, Jean-Christophe: On state merging in grammatical inference: A statistical approach for dealing with noisy data. In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003, S. 688–695

[sta] *Statista*. https://de.statista.com/statistik/daten/studie/585883/umfrage/anteil-der-smartphone-nutzer-in-deutschland/, Abruf: November 2018

[Tes] *GitHub*. https://github.com/tesseract-ocr/tesseract/blob/442b5b7/dict/trie.h#L192, Abruf: Dezember 2018