UNIVERSITÄT KOBLENZ · LANDAU

Design und Implementierung eines Business Process Modeling Recommender Systems auf Basis probalistischer Endlicher Automaten

BACHELORARBEIT

ZUR ERLANGUNG DES GRADES BACHELOR OF SCIENCE IM STUDIENGANG INFORMATIK

vorgelegt von

Tim Schneichel

[215101171]

Koblenz, im März 2019

Erstgutachter:	Prof. Dr. Patrick Delfmann (Institut für Wirtschafts- und Verwaltungsinformatik, FG Delfmann)
Zweitgutachter:	M.Sc. Christoph Drodt (Institut für Wirtschafts- und Verwaltungsinformatik, FG Delfmann)
Betreuer:	M.Sc. Christoph Drodt (Institut für Wirtschafts- und Verwaltungsinformatik, FG Delfmann)

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden. ☒ ja ☐ nein

. . . . . .

(Kruft, 25. März 2019) (Unterschrift)

Kurzfassung

Ziel dieser Arbeit ist es, ein Recommender System (RS) für Geschäftsprozesse zu erstellen, das auf dem bestehenden ProM-Plug-in RegPFA aufbaut. Um dies zu gewährleisten, soll zunächst eine Schnittstelle geschaffen werden, welche die von RegPFA erstellten probabilistischen endlichen Automaten (PFA) im tsml-Format zu einer erweiterbaren Datenbasis zusammenfassen kann. Anschließend soll ein Java-Programm geschrieben werden, das mithilfe dieser Datenbasis zu einem gegebenen Teilprozess die wahrscheinlichsten Empfehlungen für das nächstfolgende Prozesselement angibt.

Abstract

The goal of this thesis is to create a recommender system (RS) for business processes, based on the existing ProM plugin RegPFA. To accomplish this task, firstly an interface must be created that sets up and expands a database receiving probabilistic finite automata (PFA) created by RegPFA in tsml format as input. Secondly, a Java program must be designed that uses said database to recommend the process elements that are most likely to follow a given sequence of process elements.

Contents

Index of figures

Index of tables

Index of abbreviations

BPM ..... *Business process management*
EOP..... *End of process*
PFA .....*Probabilistic finite automaton*
RS .....*Recommender System*
SPE .....*Sequence of past events*

1 Introduction and Motivation

Given the recent rise of importance of big data analytics (see Chen et al. 2012 p.1), it is not surprising that process mining, which refers to “the development of tools and methods to generate insight based on event data collected during the execution of a business process” (see Breuker et al. 2016 p.2) is also surging in importance. This stems from the fact that process mining links big data analytics to business process modeling through transforming event logs into process models (Vera-Baquero et al. 2013 p.1).

At first, process mining aimed to gather insights by looking at the past, thereby performing retrospective analysis (Breuker et al. 2016 p.2). Recent approaches also implement predictive analytics, monitoring both the past and the present to assume future behavior of processes (Breuker et al. 2016 p.2). Due to participating in both a project practical as well as an undergraduate seminar in the broad field of business process management (BPM) during 2018, I had already developed an interest in process mining and process modeling. In combination with the growing need for analytic software in today’s corporate world, I was convinced that writing my bachelor’s thesis in this field of work would be a great way to obtain skills and knowledge relevant to most modern companies.

This bachelor’s thesis aims to perform predictive analysis by designing and implementing a RS for business processes using probabilistic finite automata (PFA) generated by the process modeling software RegPFA. Since recommendations cannot be given without collecting process data first, the following goals can be determined:

1. Developing a program capable of reading textual representations of PFAs generated by RegPFA and combining them to a singular database from which recommendations can be drawn.
2. Designing and implementing a RS that uses said database and generates predictions for the next event given a sequence of previously occurred events as input.

To fulfill these goals, reasonable knowledge of both RegPFA and RS’s in general is required. Therefore, the following two chapters provide a rough overview of RS’s and PFAs in general as well as RegPFA in particular. Chapter 4 then explains the steps taken to reach the goals outlined above and offers a detailed

explanation of the two components of the RS designed in this thesis, which will further be referred to as RegPFA RS. In chapter 5, several testcases are described to determine the efficiency and accuracy of RegPFA RS. The results of those tests are further discussed in the conclusion at the end of this document.

2 Recommender Systems

RS, first introduced in the mid-1990s (Portugal et al. 2015 S. 1), are omnipresent in modern day life. From YouTube (Covington et al. 2016 S. 1) to Amazon (Smith et al. 2017 S. 1) to Netflix (Gomez-Uribe et al. 2016 S. 1), lots of widely used websites that offer large selections of products point their users towards those that are most likely to interest them using RS. While that goal is similar for most RS, several approaches exist to determine which product constitutes the best recommendation for a given user. The three most common approaches collaborative filtering, content-based filtering and hybrid filtering (Portugal et al. 2015 S. 2) will be explained briefly in the following sections.

2.1 Collaborative Filtering

Essentially, collaborative filtering refers to the practice of generating recommendations for a certain user by finding other users with similar interests and recommending items that they've already purchased or liked. This can be achieved by applying a nearest-neighbor method to a matrix of ratings. (see Felfernig et al. 2007 S. 1) Additionally, characteristics of the users themselves such as age or gender may be included to calculate the similarity of two users (see Portugal et al. 2015 S. 2). The left side of Figure 1 describes this approach graphically. Collaborative filtering is meant to simulate human behavior since people are likely to watch a TV show or read a book that their friends recommend to them (see Shani et al. 2005 S. 4). The drawbacks shown in such methods include cold-start, i.e. inability to make recommendations before sufficient user data was collected, sparsity of data in general and scalability problems (see Isinkaye et al. 2015 p.3).

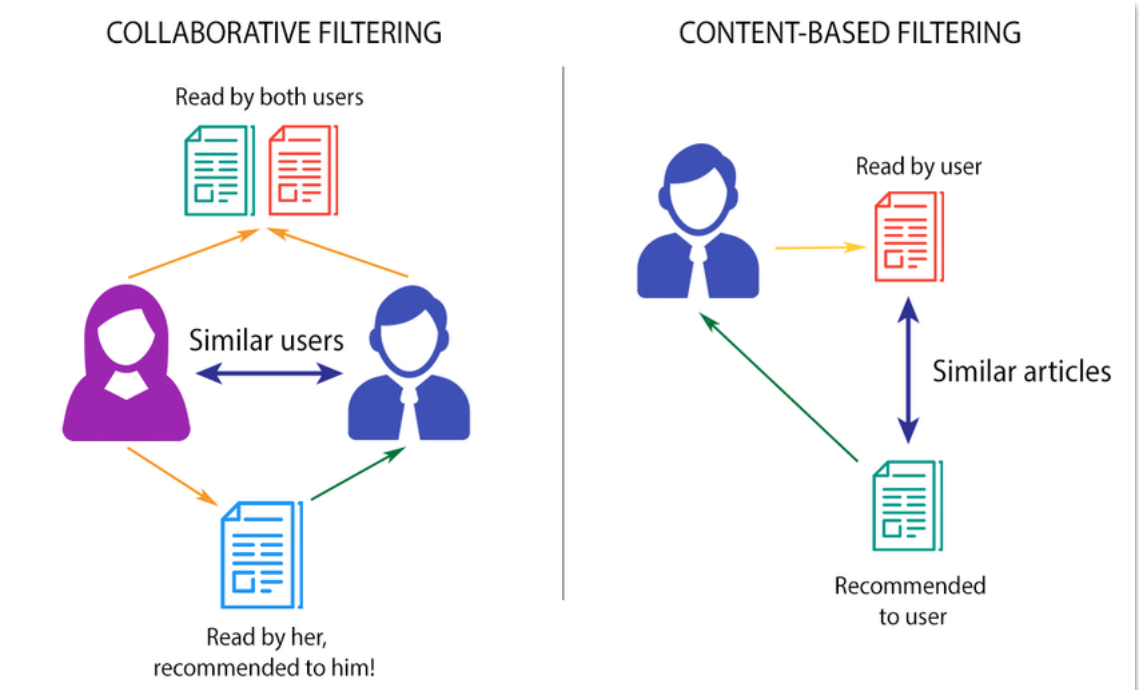

Figure 1 - Collaborative vs. content-based filtering. Source: Tondji 2018 p.11

2.2 Content-based filtering

While collaborative filtering aims to make recommendations based on similarities between users, the content-based approach to RS's focuses on similarities between items. For this method to be used, items must be defined by certain features, e.g. genre, actors and director of movies, that are used to calculate the similarity between them. (Portugal et al. 2015 S. 3) A user then receives recommendations for items that are most similar to those they already acquired in the past (Shani et al. 2005 S. 4). This process is illustrated on the right side of Figure 1.

Problems with content-based filtering techniques include limited content analysis, i.e. difficulties determining certain features correctly or too many possible values for features, as well as overspecialization and sparsity of data (see Isinkaye et al. 2015, p.3).

2.3 Hybrid filtering

As the name suggests, hybrid filtering combines metrics of both collaborative and content-based filtering to generate recommendations. The goal of hybrid filtering is to create a single RS as a combination of multiple filtering techniques to

diminish the impact of their shortcomings while taking advantage of their strengths. (see Isinkaye et al. 2015 p.3)

3 RegPFA

Considering this thesis is heavily reliant on the predictive process modeling technique RegPFA, which is short for “regularized probabilistic finite automata”, an overview of the program must be provided. First, the general functionality of the two RegPFA components, the RegPFA Predictor and the RegPFA Analyzer as shown in Figure 2, will be explained. A short overview of PFAs as well as a detailed presentation of both components of RegPFA is then given in the following chapters.

The Predictor receives an event log, i.e. a collection of sequences of events, as input and uses a learning algorithm to derive a probabilistic model representing the data from the event log. Subsequently, this model can be used to either make predictions about the outcome and behavior of present events when combined with corresponding data or it can be used as input for the Analyzer. In the latter case it is then transformed into an easily readable visualization in the form of either a Petri net or an automaton. (see Breuker et al. 2016 p.8)

3.1 Probabilistic finite automata[1]

Assuming at least a light familiarity with finite automata, the mathematical definition of a PFA is given here without going into detail. Some additional comments are provided however to indicate how certain mathematical constructs are used regarding either business process modeling in general or RegPFA in particular. For the uninitiated reader, Figure 3 at the end of this subchapter provides an illustration of a simple PFA, accompanied by a simplified explanation.

A PFA is a tuple A = (Q_A, Σ, δ_A, I_A, F_A, P_A), where:

Q_A is a finite set of states

Σ is the alphabet, i.e. a set of transition names

[1] All mathematical formulas and definitions in this section are based on Vidal et al. 2005 pp.7-8.

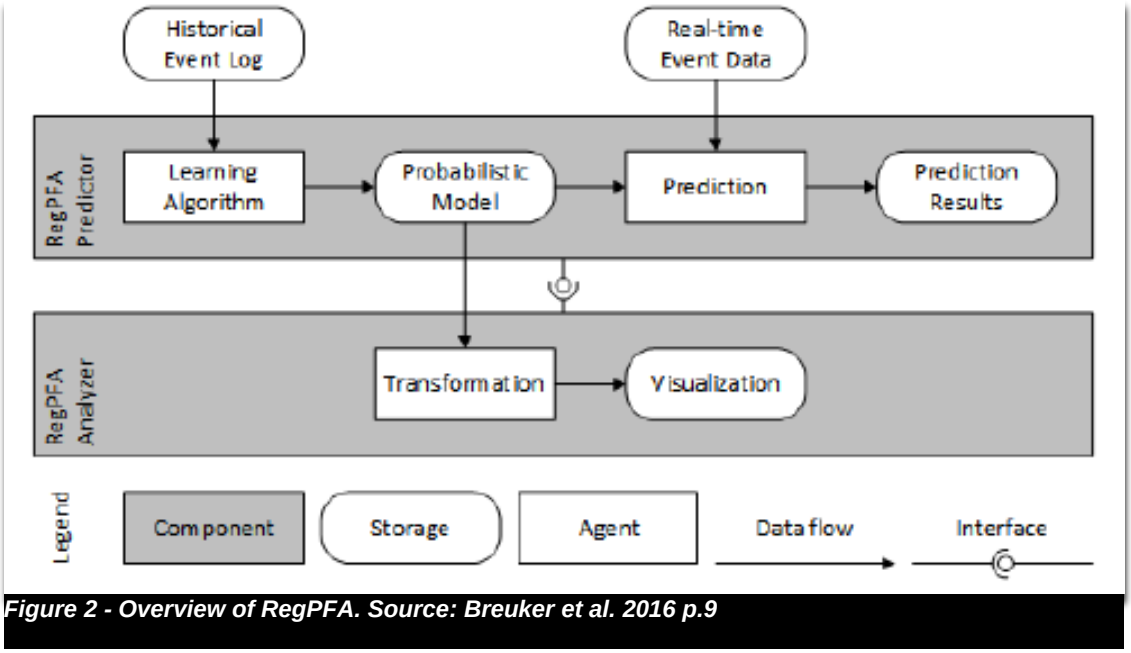

Figure 2 - Overview of RegPFA. Source: Breuker et al. 2016 p.9

$\delta A \subseteq Q_A \times \Sigma \times Q_A$ is a set of transitions, where each transition is defined by the combination of its start state, its end state and its transition name

$I_A: Q_A \rightarrow \mathbb{R}^+$are the initial-state probabilities

$F_A: Q_A \rightarrow \mathbb{R}^+$are the final-state probabilities

$P_A: \delta A \rightarrow \mathbb{R}^+$are the transition probabilities

It further holds that

$$\sum_{q \in Q_A} I_A(q) = 1$$

and

$$\forall q \in Q_A: F_A(q) + \sum_{a \in \Sigma, q' \in Q_A} P_A(q, a, q') = 1$$

Those restrictions state that the sum of all probabilities to start in a given state q is 1 and that the sum of the probability to end in a given state q plus the probability to leave state q to any other state q' is 1 as well. As explained in more detail in chapter 3.2, PFAs generated by RegPFA are modified to only contain a single start and end state. Figure 3 shows an exemplary PFA. There is only one initial state, namely q_0 with $I_A(q_0) = 1$. Each state is a possible final state though with probabilities to end in this state instead of transitioning to another state noted within the circle denoting said state.

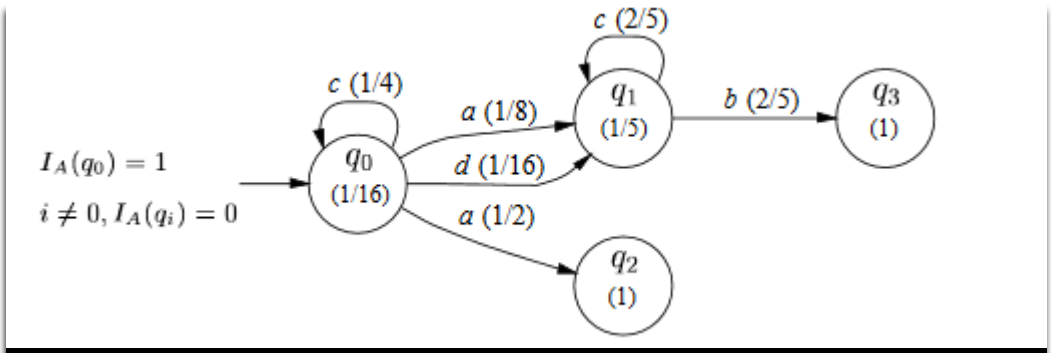

Figure 3 - Exemplary PFA. Source: Vidal et al. 2005 p.8

PFA in graphical representation can be read similarly to control flow graphs. Since q_0 is the sole initial state, reading always starts there as indicated by the arrow from the left. The outgoing arrows from q_0 indicate which state it is possible to move towards and are labeled with the name of transition as well as the probability of the transition. E.g. it is possible to move from q_0 to itself with probability 1/4 using a transition labeled “c.”

In the context of this thesis, transitions represent events taken from an event log. Therefore, visually moving through a PFA implies that the sequence of transitions that are used to move from state to state correspond to a possible sequence of events in a given event log. E.g. the sequence of (c, a, c, c) constitutes a valid way to transition through the PFA above, ending at q_1.

3.2 RegPFA Predictor

Since the RS explained in this thesis works with the visualization derived by the RegPFA Analyzer, it is sufficient to discuss the properties of the probabilistic model used as a basis for the Analyzer. Taking a standard PFA as a basis for the model, some constraints are set to suit the Predictor to BPM (see Breuker et al. 2016 p.8).

Generally, a PFA does not necessarily have only one start or end state. A critical goal of RegPFA is to create a visualization that can easily be understood without comprehensive knowledge of probabilistic models (see Breuker et al. 2016 p.3). The first modification therefore consists of altering the model to only include a single start and end state, which additionally results in the PFA closer resembling workflow nets, which also have fixed starts and ends (see Breuker et al. 2016 p.8).

Furthermore, it is also required to impose a constraint regarding the estimation of probabilities in the model. RegPFA uses a parameter estimation approach to realize grammatical inference, namely a maximum likelihood estimator. On its own, this method could yield widely inaccurate results for incomplete or insufficient data due to overfitting. However, as a counteract the Predictor adds a modification based on Bayesian regularization to the maximum likelihood estimator. Essentially, this means that pseudo-observations of all events at all states are added before considering the actual observations from the input event log. (see Breuker et al. 2016 pp.8-10)

To illustrate this process, imagine randomly picking a single card from a standard sized deck of cards 1000 times which turns out to be the ace of spades each time. The maximum likelihood method infers that the deck most likely is made up of 52 aces of spades, which seems plausible. However, if we only pick a single card once and it turns out to be the ace of spades, maximum likelihood would still infer that the deck is most likely made up of 52 aces of spades, which contradicts the instinctive assumption that the standard sized deck indeed contains the standard set of cards.

In a Bayesian framework, it is possible to apply some sort of ground assumption to the probability distribution to mathematically implement the belief that probabilities close to one or zero are unlikely (see Breuker et al. 2016 p.9). For instance, if we apply the assumption that there are 52 distinct cards in the deck and the probability to draw each one of them is 1/52, meaning we have 52 pseudo-observations of card draws, drawing the ace of spades once changes the expected probability of drawing the ace of spades from 1/52 to

$$p(ace\ of\ spades) = \frac{1+1}{52+1} = \frac{2}{53} \approx 0{,}04$$

While changing the probability of any other card *c* that is not the ace of spades to

$$p(c) = \frac{1+0}{52+1} = \frac{1}{53} \approx 0{,}02$$

However, if we again assume to draw the ace of spades 1000 times in a row:

$$p(ace\ of\ spades) = \frac{1+1000}{52+1000} = \frac{1001}{1052} \approx 0{,}95$$

$$p(c) = \frac{1+0}{52+1000} = \frac{1}{1052} \approx 0{,}001$$

It is easy to see that with this modification it is still possible to depict extreme results when given sufficient data, but predictions are more resistant to insufficient or incomplete data. For the RegPFA Predictor, a Dirichlet distribution, i.e. a continuous, multivariate probability distribution, with symmetric priors of starting value

$$hp = 1 + \frac{n}{K * (1 + E * (1 + K))}$$

for each Dirichlet parameter *hp* was chosen. In this formula, *n* refers to the amount of pseudo-observations, *K* refers to the amount of states of the PFA and *E* to the number of distinct events in the event log. (see Breuker et al. 2016 pp.9-10)

A maximum likelihood estimate is then determined by running an expectation maximization algorithm. After choosing initial parameters randomly, it switches between executing an expectation step, where current values of parameters are used to derive a function of the log likelihood and a maximization step, where new values for the parameters are calculated which maximize the log-likelihood of the function derived in the previous expectation step.The algorithm converges once the rate of improvement falls below a certain threshold set by the user. This process is repeated multiple times as the expectation maximization algorithm finds local optima depending on the initial randomly set parameters. (see Breuker et al. 2016 pp.10-11)

3.3 RegPFA Analyzer

To recap, the main task of the RegPFA Analyzer is to transform a probabilistic model generated from running the RegPFA Predictor into an easily readable visualization. The key phrase in the previous sentence is “easily readable” since an important design goal of the RegPFA creators was to provide a visual output that can be understood by users inexperienced in probabilistic modeling. Since PFAs are created from event logs, generally those of business processes, achieving said goal is beneficial since it enables experts in the respective area of business to work with RegPFA’s output. (see Breuker et al. 2016 p.12)

A direct consequence of this aim is to cut transitions with zero probability from the visualization. However, as explained in the prior subchapter, Bayesian regularization is used, which means transitions will not have zero probability value

except for few exceptions. This leads to a conflict of interest: On the one hand, the non-zero probabilities caused by Bayesian regularization are necessary to avoid overfitting. On the other hand, keeping all probabilities in the graphic leads to a PFA with *K* states and K^2E transitions, with *E* denoting the amount of unique transition names, which is contrary to the goal of the graphic being easily readable. (see Breuker et al. 2016 p.12)

The compromise made by the RegPFA team was to impose a probability threshold ε for transitions, meaning all transition probabilities of value below ε are disregarded for the visualization. It was defined relative to the value of

$$p_{equal} = \frac{1}{K * E}$$

which constitutes the probability of any transition from either state to another assuming all transitions are equally likely. As the sum of all transition probabilities from a single state will always be 1, this directly implies that if some probabilities to leave a given state towards another state increase, others must decrease. (see Breuker et al. 2016 pp.12-13)

The RegPFA Analyzer requires the user to input a pruning ratio *pr* that will be used to determine the actual value of ε according to the formula

$$\varepsilon = \frac{1}{pr} * \frac{1}{K * E}$$

Therefore, transitions will be cut if they are *pr* times as unlikely as any transition from a state when no further information was provided. Naturally, if all transitions leading into a certain part of the PFA are cut due to their probability being below ε, all states and transitions in that part are disregarded as well. This is illustrated in Figure 4, where the left side depicts a PFA generated with $\varepsilon = 0.1$ and the right side depicts the same PFA generated with $\varepsilon = 0.2$. (see Breuker et al. 2016 pp.12-14)

Automata which have been pruned in that matter may already fulfill the requirement of being easy to read. In fact, the intermediate visualization of a pruned PFA is used as input for the RegPFA RS described in this thesis. It can however be transformed into an equivalent Petri net to further ease the interpretation of the graphic. As neither the process of generating the Petri net representation nor an understanding of Petri nets in general is required to

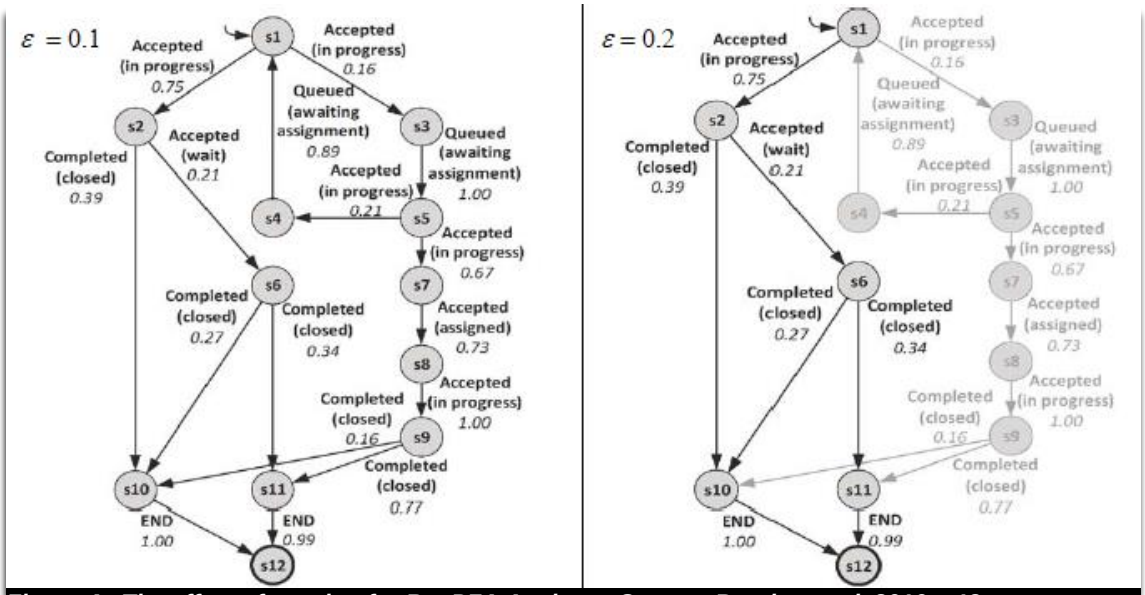

Figure 4 - The effect of pruning for RegPFA Analyzer. Source: Breuker et al. 2016 p.13

comprehend the following chapters, no further details will be provided here.[2] (see Breuker et al. 2016 p.14)

4 RegPFA Recommender System

After establishing a general knowledge of both RS's and RegPFA, the design of RegPFA RS may be discussed. In subchapter 4.1, several requirements are determined that need to be fulfilled by the final RS. Afterwards, the general functionality of the RegPFA RS two components, the Reader and the Recommender, is explained in subchapters 4.2 and 4.3. Special attention is paid to the fulfillment of the previously outlined design requirements.

Since the following chapters contain several examples of simple PFAs and transition sequences within those, a proper notation must be introduced for certain elements. Single transitions will be surrounded by parentheses, e.g. *(A)*, sequences of transitions will be surrounded by square brackets and separated by commas, e.g. *[A, B]*. The sequence of past events that constitutes the central user input of this RS will be referred to as *SPE* from now on.

[2] The inclined reader may find additional information about this process in "Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting (van der Aalst et al. 2010)

4.1 Preliminary considerations

When designing a RS, it is necessary to consider the type of data one is working with. In this section, we will attend to the PFA. Considering the formal definition of a PFA given in chapter 3.1, that means we have a set of states, an alphabet and three sets of probabilities: the probabilities to transition from one state to another, the probabilities to start in any state and the probabilities to end in any state. Conveniently, PFAs generated by RegPFA are modified to only contain a single start and end state, rendering the latter sets of probabilities obsolete.

Since RegPFA solely relies on event logs to create PFAs and does not receive other inputs, there is no user specific data available aside from potential insight into process structure, which will be displayed in the PFA anyway. This strongly implies the use of content-based filtering for the RS. Additionally, in a potential real-world context, corporations making use of process mining techniques like RegPFA may not be willing to share internal information concerning their BPM.

.

4.1.1 Fundamental requirements

Now that it has been established that only a content-based approach is appropriate, we can define requirements for the RS. Some fundamental requirements need to be defined independent of possible content, namely:

[R1] – Recommendations based on longer subsequences of *SPE* are always rated better than those based on shorter subsequences.

This is an obvious requirement whose main purpose is to express the importance of context. It is best explained as an analog to text-based RS: When entering the phrase “It’s raining cats and”, any decent RS will recommend “dogs” next. However, just entering “and” will yield other recommendations than “dogs” most likely. In terms of business processes, finding a transition that follows the exact *SPE* of *[A, B, C]* provides far more valuable information than finding a transition that follows *[C]*.

[R2] – The last element of *SPE* is the primary factor for recommendations.

As mentioned above, context is of utmost importance when making recommendations. Ultimately, the last element of *SPE* provides the most recent context, i.e. the event that transpired directly before the point from which a

recommendation must be made. While other approaches, e.g. choosing the overlap of events preceding a certain event with *SPE* as primary factor, might also be suitable, it was decided to value the most recent context highest.

4.1.2 Additional requirements

Requirements for the RS can also be derived by considering different kinds of contents and inputs and deciding how the RS should behave in those cases. For instance, consider PFA_1 illustrated in Figure 5. If the database only contained PFA_1 and *SPE = [A]*, it is obvious that *(B)* should be recommended as the next event. However, there are several possible values for *SPE* that hold less obvious implications.

1) SPE = [B]:

A transition that follows *[B]* does not exist in the database, a transition *(B)* does exist in the database though. There are two possible ways to handle this case:

- Conclude that no recommendations can be made
- Assume that the absence of a following transition to a known transition requires a special recommendation independent of the database, e.g. *(end of process),* further abbreviated as *(EOP)*

Since identifying the end of a process provides valuable information and is part of the RegPFA team's main goal of "[predicting] future behavior of business processes" (see Breuker et al. 2016 p.2), the second option was chosen leading to the third requirement for the RS:

[R3] – Recommend *(EOP)* when *SPE* leads to the final state

2) SPE = [ ]:

If the input list is empty, it is impossible to know which transition follows next. There are three possible ways to handle this case:

- Conclude that no recommendations can be made
- Recommend the transitions that most often originate from the start state
- Recommend the transitions that most often occur in the database, disregarding context

It seems intuitive to choose the second option, mirroring the intended behavior of

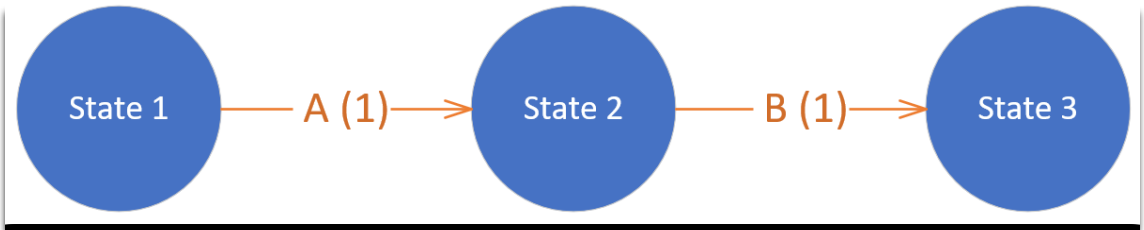

Figure 5 – PFA_1 with three states and two transitions. Source: self-made

the RS concerning **R3**. The third option also seems reasonable at first glance, since finding the most frequent transition appears to be equivalent to finding the most often frequently transpiring event, which provides valuable information. This assumption is incorrect though.

Consider PFA_2 illustrated in Figure 6. If our database was extended to include it as well, *(B)* would be the most frequently occurring transition with a count of 2 and a summed probability of 1.1. However, we do not know how many process instances for each PFA are performed. E.g. it is possible that PFA_1 is associated with 10 process instances while PFA_2 is associated with 100 process instances, leading to 10 instances of *(A)*, 20 instances of *(B)*, 90 instances of *(D)* and 100 instances of *(C)* on average. Therefore, the second option was chosen, which implies the fourth requirement:

[R4] – Recommend transitions from starting states if *SPE* is empty

3) SPE = [C]:

The database, consisting again of just PFA_1, contains no transition *(C)*, therefore a proper recommendation cannot be made. There are two possible ways to handle this case:

- Conclude that no recommendations can be made
- Recommend the transitions that most often occur in the database, disregarding context

Choosing the last option strongly implies to the user that the recommendation was made directly based on information gathered from the database. Since no such information for an *SPE* of *[C]* exists, the first option seems optimal. However, not making any recommendation implies that the database does not contain any helpful information to make a prediction at all, which is also not entirely true. Transitions appearing across multiple PFAs or multiple times within a single PFA indicate events that are important for several different processes and can be identified within the database.

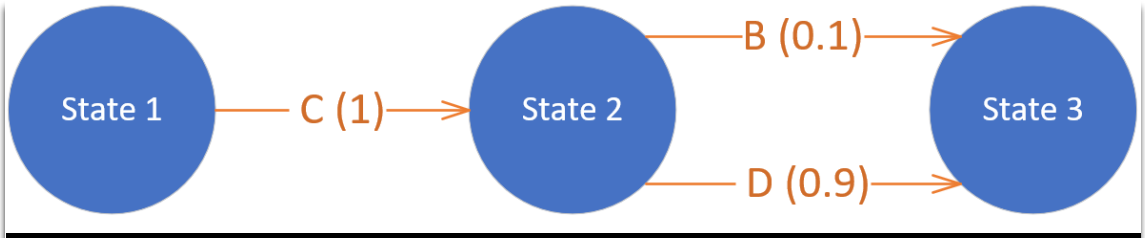

Figure 6 – PFA_2 with three states and three transitions. Source: Self-made

As a compromise, it was therefore decided to recommend the transitions which most often occur in the database while displaying a warning that those recommendations are not based on *SPE*. This will always be the case if the last transition in *SPE* is not part of any PFA in the database. The consequential requirement is as follows:

[R5] – Recommend most frequent transitions when the database does not contain the last element of *SPE* while indicating those recommendations are not based on *SPE*.

Consider a database consisting of both PFA_1 from Figure 5 as well as PFA_3 from Figure 7. With *SPE = [B]*, *(EOP)* as well as *(E)* are obvious events to be recommended. The probability of *(E)* in PFA_3 is 1, the probability of *(EOP)* in PFA_1 is 1 as well, considering there are no possible transitions from state 3. Although at first glance it may seem that implies *(E)* and *(EOP)* should be recommended with the same probability, the paths that lead to the transition labeled *(B)* in both PFAs needs to be considered as well. For PFA_1, there is only a single path leading to state 2 from which *(B)* originates with probability 1. For PFA_3, there is also just one path leading to the origin state of *(B)*, but the probability of that path is just 0,5.

As mentioned above, no data regarding association with specific process instances is provided, therefore it's impossible to conclude which PFA represents more actual business processes. Lacking actual data, it is assumed that each PFA represents the same actual amount of business processes. Therefore, applying the assumption as well as the given information that the last transpired event was *[B]*, it is twice as likely for the process to occur in PFA_1 compared to PFA_3. This implies it is twice as likely that *[B]* is followed by *(EOP)* compared to *(E)*, expressed in the final requirement:

[R6] – Probabilities of the origin states of *SPE* in each PFA must be considered, not only the probability of elements in *SPE*.

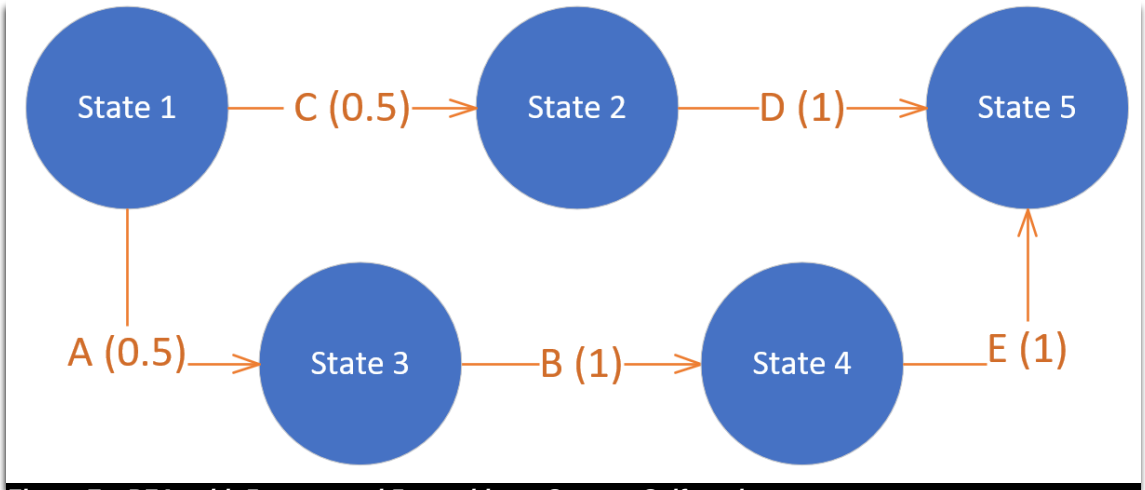

Figure 7 – PFA$_3$ with 5 states and 5 transitions. Source: Self-made

4.2 Reader

The main task of the RegPFA RS Reader is to create and update a database containing several PFAs generated by RegPFA. Since textual representations of them exist as files in tsml format, it can be accomplished by writing a straightforward parser which collects relevant information and disregards anything else. Additionally, to guarantee efficiency of the Recommender, data needs to be prepared in a way that facilitates the generation of recommendations, especially regarding the requirements outlined in the previous chapter.

Tsml files can be generated by using the ProM plugin RegPFA to create a transition system from a log file.[3] Once this is completed, the newly generated transition system can be viewed in the ProM workspace. After clicking on “Export to disk” as shown in Figure 8, making sure to select the tsml file format in the save menu, the transition system can be converted to tsml format. A tsml file is essentially split in two sections: First, all states of the transition system are listed by name, then all transitions including their respective source and target states are listed. The left side of Figure 9 illustrates an exemplary state in tsml notation while the right side showcases an exemplary transition.

The parser starts by going through the state part of the file, creating all states with only their ID currently known. Once the transition part of the document is reached, each transition is created and named according to its label, then linked to the

[3] A detailed guide for this process can be found at https://em.uni-muenster.de/wiki/Mining_with_RegPFA_algorithm

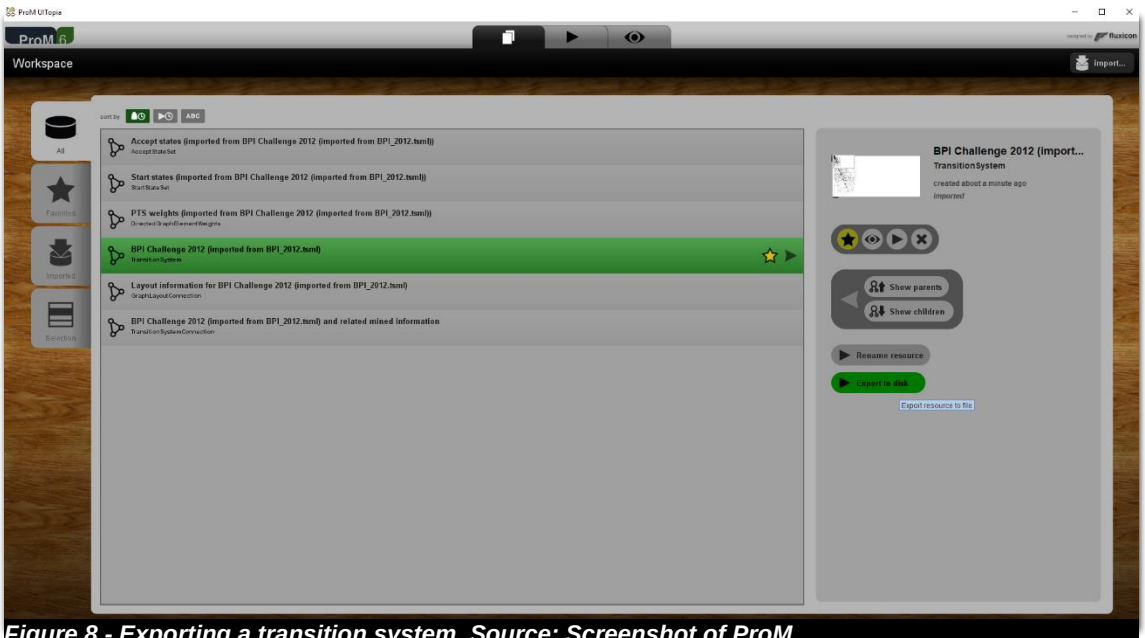

Figure 8 - Exporting a transition system. Source: Screenshot of ProM

```
<state id="state1">
<name>
<text>
s0</text>
</name>
<graphics>
<position x="866.5" y="1555.5"/>
<dimension x="50.0" y="30.0"/>
<fill color="#c0c0c0"/>
</graphics>
</state>
```

```
<transition id="transition136" source="state23"
target="state23">
<name>
<text>
W_Valideren aanvraag(SCHEDULE) [0.4078]</text>
</name>
<graphics>
<position x="625.0" y="158.0"/>
<position x="647.5" y="150.5"/>
<position x="670.0" y="158.0"/>
</graphics>
</transition>
```

Figure 9 - Exemplary notation of states and transitions in tsml files. Source: Self-made graphic based on the transition system of the BPI Challenge 2012 (van Dongen 2012) generated by RegPFA

respective source and target states. Additionally, the probability of said transition to occur from its source state is stored. Considering references from an object to another object in Java are not bidirectional by default, a reference from both source and target state to the transition connecting them must be created as well. Otherwise the process of finding transitions from or to a given state, which is needed to fulfill **R1** and **R6,** would be slow and inefficient.

Those links are then used to calculate the probabilities to land on each state when randomly traversing the PFA. This is performed by first determining the start state of the PFA and then starting a recursive function from it that follows every possible sequence of transitions in the PFA and adds the current probability of the sequence to each state after it reaches it. However, if the sequence lands on a state that was already visited over the course of the sequence, i.e. a circle was found within the PFA, the recursion stops, and the sequence's probability is not

added to the state yet again. While a pseudocode representation of this function is provided in Figure 10, it is reasonable to also include a mathematical notation for the calculation of state probabilities.

SS denotes the starting state of the PFA. *(s_i, s_j)* denotes a transition from state s_i to state s_j and *p(s_i, s_j)* the probability of the transition. *[($s_{i(1)}$, $s_{j(1)}$), ($s_{i(2)}$, $s_{j(2)}$), … ($s_{i(n)}$, $s_{j(n)}$)]* denotes a transition sequence from state $s_{i(1)}$ to state $s_{j(n)}$ if

$j(k) = i(k+1)\ \forall\ 1 \leq k \leq n-1$. A transition sequence is called non-circular if $j(k) \neq j(l)\ \forall\ k, l\ \in \mathbb{N}$. The set of all non-circular transition sequences is called *NC*. Finally, the probability of a state *S* is defined as

$$p(S) = \sum_{\substack{[(s_{i(1)},s_{j(1)}),(s_{i(2)},s_{j(2)}),\dots(s_{i(n)},s_{j(n)})]\epsilon NC \\ \wedge s_{i(1)} = SS \wedge s_{j(n)} = S}} \prod_{k=1}^{n} p(s_{i(k)}, s_{j(k)})$$

For bigger PFA, this calculation is not feasible. While PFAs are unlikely to be extremely interconnected, the amount of unique non-circular paths within a PFA with (n+2) states - with the start state and the end state being fixed irrelevant in terms of state probability - can be higher than 2 * n! in total. E.g. a PFA with 30 states excluding start and end could already require more than $5*10^{32}$ single operations to calculate state probabilities. Therefore, if the amount of states excluding start and end within a given PFA is higher than 20, all state probabilities are set to 1, rendering them irrelevant for further calculations. Recommendations in PFAs of that size are therefore slightly less accurate.

A basic recommender could already be developed using the state and transition data outlined above but fulfilling **R5** requires a list of all transitions grouped by their label and sorted by their frequency of occurrence. Since creating this list whenever a new recommendation is required would be resource intensive, it is stored as part of the database as well as updated and sorted whenever a new transition system is read. Even though merge sort is set as the standard sorting algorithm, an alternative sorting algorithm named “divide sort” is also included albeit not being used by default.

While merge sort assumes an unsorted list of *n* elements and performs a divide and conquer on the whole list, divide sort assumes that the left part of the list, i.e.

```
setStateProbabilities (S, LS, P)
Inputs:
S: The current state
LS: List of states that were already visited
P: Probability of the current transition sequence
Output:
Nothing
Start
set Probability of S = Probability of S + P
Add S to LS
for (Transition T originating from S)
        if not (LS contains target state TS of T)
                P = P * probability of T
                setStateProbabilities (TS, LS, P)
        end if
end for
```

Figure 10 - Pseudocode of state probability algorithm. Source: Self-made

the transitions with high frequencies, is already sorted. It identifies the index of the last sorted element i, then divides the list into a sorted left part and an unsorted right part, on which merge sort is performed. Afterwards, the already sorted left part and the newly sorted right part are merged. Thereby, merge sort only needs to be performed on a list with *n-i* elements instead of a list with *n* elements. Assuming *i* to be marginally smaller than *n*, divide sort has a lower total runtime and is hence preferable to merge sort.

For ease of use, tsml files only need to be moved to the folder labeled “data” within the program folder. Once started, the Reader then analyzes the content of all tsml files within the data folder and moves them to the archive folder afterwards, overwriting files with the same name. The pseudocode of a single iteration of this process is shown in Figure 11.

4.3 Recommender

Generating recommendations is set up as a multi-step process: First, the user input consisting of *SPE*, a maximum number of required recommendations and a weight factor for subsequences is processed. Using the database created by running the Reader at least once, the Recommender then iterates over all stored PFAs to find out in which of them the *SPE* can occur. For each suitable PFA, it is

Inputs:
TF: tsml file containing textual representation of a PFA
DB: Database filled with PFA
LT: list of all transitions currently known to the database grouped by name and ordered by frequency
Outputs:
DB and *LT* are altered
Start
Generate empty new PFA *P*
Repeat
    Generate new state for *P*
Until (All states from *TF* were generated)
Repeat
    Generate new transition *T* for *P*
    Set label and probability of *T*
    Link *T* to source and target state
    Link source and target state to *T*
    Increase frequency of *label(T)* in *LT* by 1
Until (All transitions from *TF* were generated)
Add *P* to *DB*
Merge sort *LT*

Figure 11- Pseudocode of RegPFA RS Reader. Souce: Self-made

then determined which states *SPE* can lead to and finally which transitions are possible from that state. Those transitions are saved as possible recommendations.

If not enough recommendations were found, the first element of *SPE* is cut and the recommendation process begins anew until either enough unique recommendations were found or *SPE* contains no more elements. Recommendations containing identical transition names are then grouped and merged into a single recommendation. To achieve this, the probability of every single recommendation is adjusted by a factor dependent on the number of events in *SPE* used to generate it as well as the weight factor set by the user. A detailed explanation of all steps outlined above will be given in the following subchapters.

4.3.1 User input

In order to simplify repeated runs of the RegPFA RS Recommender, all user input is entered into a text file located in the data folder and automatically read with each program start. An exemplary content of this file is depicted in Figure 12.

Naturally, the user is required to define *SPE*, which constitutes the single most important parameter for the RS. Equivalent to the notation used in this thesis, this is accomplished by entering a comma-separated series of transition names within square brackets.

Additionally, the user may specify the amount of recommendations that he wishes to receive as well as a weight factor for subsequences. While the first parameter is self-explanatory, the second one requires further explanation which will be provided in chapter 4.3.3.

4.3.2 Identifying possible recommendations

Determining which transitions are most likely to follow a given *SPE* is the key task of the RegPFA RS and unsurprisingly the most complex one as well. Figure 13 provides an incomplete pseudocode of the recommendations process, leaving out the task of filling up the list of recommendations *LR* with the most frequently occurring transitions in the whole database when *SPE* does not occur in any PFA. The implementation of this subtask is comparably trivial, but nonetheless needed to fulfill **R5**. Additionally, the same algorithm is used to fill up *LR* if some, but not at least *AR* "real" recommendations, i.e. transitions directly following *SPE* or a subsequence thereof, could be found.

The set of recommendations generated in this way will further be called R_F. To prepare for the ranking process outlined in chapter 4.3.3, the probability *p(R)* for each recommendation *R* in R_F is set to

$$p(R) = \frac{f(R)}{\sum_{R' \in R_F} f(R')}$$

with *f(R)* denoting the frequency of transitions with the same label as *R* over all PFAs in the database.

The Recommender begins by checking whether *SPE* is empty. If that is the case, all starting states from all PFAs in the database are possible end states for *SPE* as outlined in **R4**. Conveniently, due to modifications made to the PFAs generated by RegPFA as explained in chapter 3.2, there is only a single starting state per PFA which therefore has a fixed probability of 1. It is hence sufficient to set the probability of any recommendation of an outgoing transition from a starting state to the probability of the transition itself.

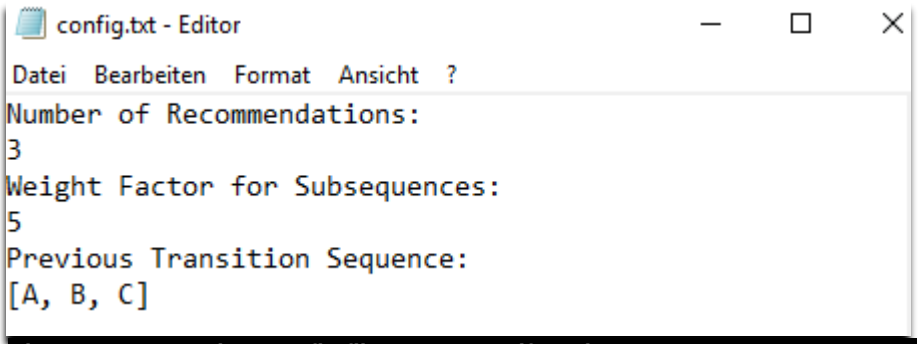

Figure 12 - Exemplary config file. Source: Self-made

Otherwise, it needs to be determined for all PFAs in the database whether they contain *SPE* as a possible sequence of transitions, not necessarily originating from the start state. Keeping **R1** as well as the user specified maximum amount of recommendations in mind, this is accomplished by iterating over all PFAs repeatedly, removing the first element of *SPE* after each iteration, until either *SPE* is empty or enough recommendations were found.

For the actual search for instances of *SPE* within a PFA, a backtracking algorithm was implemented. Filtering out all PFAs that do not contain all transitions occurring in *SPE* first, within the remaining PFAs a recursive function *getRecommendations* is then called for all possible target states (*CS)* of the first element in *SPE*. Since it is not only possible, but due to the nature of RegPFA highly likely that any given state is the source of multiple transitions with the same name, there are often multiple ways to traverse a PFA from *CS* according to *SPE*. All possible directions, i.e. transitions with the same name as the next element in *SPE* originating from *CS*, need to be checked. Therefore, the recursive call of *getRecommendations* is nested in a for-loop. Figure 14 shows a pseudocode representation of this function.

Due to the removal of the first element in each iteration, the last element of *SPE* will stay relevant throughout every single iteration, simultaneously fulfilling **R2**. And since the total probability of any given recommendation is multiplied by the probability of the currently considered transition *T* after each recursive call,

```
Inputs:
SPE: Series of past events
DB: Database filled with PFA
AR: Number of recommendations
Output:
A list of AR recommendations
Start
LR = List of Recommendations
If (SPE is empty)
        for (PFA in DB)
                for (Start state S in PFA)
                        for (Transition T originating from S)
                                Create new recommendation R
                                Probability of R = Probability of T
                                Label of R = Label of T
                                Add R to LR
                        end for
                end for
        end for
else
        while (less than AR recommendations in LR and SPE not empty)
                for (PFA in DB)
                        if (SPE exists in PFA)
                                for (Possible end state ES of SPE)
                                        for (Transition T originating from ES)
                                                Create new recommendation R
                                                Probability of R = Probability of T * Probability of ES
                                                Label of R = Label of T
                                                Add R to LR
                                        end for
                                end for
                        end if
                end for
                Remove first element of SPE
                end while
end if
merge recommendations
weigh recommendations
```

Figure 13 – Partial pseudocode of RegPFA RS Recommender. Source: Self-made

starting with the probability *P* to land on *CS*, **R6** is also fulfilled. Finally, **R3** can be fulfilled by creating a pseudo-recommendation labeled *(EOP)* whenever the recursive call of *getRecommendations* finds an empty *SPE*, but cannot find any transitions *T* originating from *CS*. The probability of *(EOP)* is naturally found to be

```
getRecommendations (CS, SPE, P)
Inputs:
CS: Current state in the PFA
SPE: Series of past events
P: probability to land on CS
Output:
A list of recommendations
Start
LR = List of Recommendations
If (SPE is empty)
        for (Transition T originating from CS)
                Create new recommendation R
                Probability of R = Probability of T * p
                Label of R = Label of T
                Add R to LR
        end for
        return (LR)
end if
if (CS contains transition labeled SPE [0])
        newSPE = SPE without first element
        for (Transition T originating from CS labeled SPE [0])
                newCS = target state of T
                newP = P * Probability of T
                add getRecommendations (newCS, newSPE, newP) to LR
                return (LR)
        end for
else
return nothing
```

Figure 14 - Pseudocode of backtracking algorithm. Source: Self-made

equal to *P* in the final recursive call since there is only one end state for each PFA generated by RegPFA and the probability to end the process in that state is always 1.

To prepare for the ranking of each unique recommendation, the current length of *SPE* is added to the probability of each recommendation that was generated naturally, i.e. not from an empty *SPE*. With *R* being a recommendation and *p(R)* denoting the ground probability of *R* calculated according to the code in Figure 14, that means the following calculation is performed:

$$p(R) = p(R) + |SPE|$$

This ensures that a recommendation based on a longer subsequence of *SPE* will always have a higher probability than one based on a shorter subsequence,

which is needed to fulfill **R1**. While the newly calculated values cannot yet be considered proper probabilities as they are always bigger than 1, they will be transformed into proper probabilities as explained in the following subchapter.

4.3.3 Ranking and weighing recommendations

To summarize the results of the steps taken by the Recommender up to this point, a list of several recommendations, i.e. transition names coupled with a pseudo probability value, exists. Some or even all recommendations might not be based on *SPE*, but rather on the frequency of occurrence of all transitions over all PFAs in the database. The set of those recommendations is called R_F. With *p(R)* denoting the probability of a recommendation according to the notation introduced above, it holds that

$$\forall R \in R_F: 0 \leq p(R) \leq 1$$

The set of recommendations based on *SPE* respectively will be called R_S from now on. It holds that

$$\forall R \in R_S: 1 < p(R)$$

Therefore, the probability of any recommendations in R_S is guaranteed to be greater than the probability of any recommendation in R_F.

It is important to remember at this point that while the amount of recommendations with unique labels was counted to ensure that at least *AR* unique recommendation are listed, R_S might contain multiple recommendations of the same label. This is an intended consequence of the fact that while longer subsequences of *SPE* are supposed to lead to predictions with higher probability, a ranking of predictions stemming from subsequences of *SPE* with equal lengths is still required. To introduce such a ranking, recommendations with identical labels are collected regardless of the length of *SPE* used to generated them and then merged to a single recommendation of that label.

The process of merging first requires grouping all recommendations by their respective label. Once this rather trivial task is completed, a merging function is applied to each cluster of recommendations. Figure 15 shows the pseudocode for a single application of the merging algorithm to a list of probabilities stemming from recommendations with the same label. Defining the list of all probabilities as

Inputs:
LP: List of probabilities
WF: Weight factor specified by user
Output:
A single probability
Start
HP = Highest probability in *LP*
UL = **ceil** (*HP*)
Remove *HP* from *LP*
for (Probability *P* in *LP*)
    rest = UL – HP
    *numerator = rest * P*
    *denominator = UL * (UL –* **floor** *(P)) ^ WF*
    HP = HP + numerator / denominator
end for
return *HP*

Figure 15 - Pseudocode of merging algorithm. Source: Self-made

$$LP = \{P_1, P_2, \dots, P_n\}$$

and without loss of generality assuming P_1 to be the highest probability in *LP*, the algorithm computes the total probability *TP* recursively as

$$TP(i) = \begin{cases} P_1 \; for \; i = 1 \\ TP(i-1) + \dfrac{(\lfloor P_1 \rfloor + 1 - TP(i-1)) * P_i}{(\lfloor P_1 \rfloor + 1) * (\lfloor P_1 \rfloor + 1 - \lfloor P_i \rfloor)^{WF}} \end{cases}$$

where *WF* denotes the user specified weight factor. It can be proven that *TP(i)* has an upper limit of $\lfloor P_1 \rfloor + 1$ through induction. The appendix contains the full inductive proof for the inclined reader.

The upper limit for *TP(i)* implies that a recommendation cluster of arbitrary size will never receive a probability equal to or higher than the next highest integer of the biggest probability within the cluster. As mentioned before, probabilities for all recommendations were inflated by adding the number of elements in *SPE* at the time of finding the recommendation. This imposes a limitation for each recommendation cluster's total probability. E.g. consider a recommendation cluster with highest inflated probability of 6.5. Since proper probabilities are greater than 0 and smaller than or equal to 1, the number of elements in *SPE* used to generate this probability had to be 6. The upper limit for this cluster's total probability therefore is 7, which means it can never reach the minimal total

probability of a cluster which contains at least one probability that's greater than 7. This guarantees that **R1** is always fulfilled.

Since a clear ranking of recommendations has been introduced now, recommendations may be weighed in order to obtain proper probabilities once more. The RegPFA RS obtains two weighing methods, of which the first one weighs all probabilities proportionally and the second one weighs them according to the softmax function. Denoting the set of all recommendations, i.e. unique transition labels matched with their respective cluster's total probability, as *LR* once more, the standard weighing method defines the weight of a single recommendation *R* as

$$weight\_standard(R) = \frac{p(R)}{\sum_{R' \in LR} p(R')}$$

While this weighing method is already sufficient for recommendation purposes, its drawback is that it does not accurately reflect the importance of **R1**. While recommendations gathered from longer subsequences of *SPE* will always be rated higher than those gathered from shorter subsequences, the difference between their ratings will be roughly proportional to the difference between the subsequences' lengths. The softmax function offers a way to weigh longer subsequences substantially more by defining

$$weight\_softmax(r) = \frac{e^{P(R)}}{\sum_{R' \in LR} e^{P(R')}}$$

While the standard weighing function might be preferable for certain cases, it was considered to be inferior to the softmax weighing function due to the perceived importance of **R1**. Therefore, the RegPFA RS offers no option to choose standard weighing for recommendations at this point.

5 Evaluation

To evaluate the accuracy and speed of the recommendations generated by the RegPFA RS, first several small-scale testcases and finally some larger-scale testcases were performed. Based on the requirements defined in chapter 4.1, the exemplary PFA from Figure 5 and Figure 7 were modeled in tsml format and given to the RegPFA RS Reader as input first. Afterwards, additional PFAs were created by hand to test the RS ability to handle circles within the PFAs as well as

the weighing function over a large amount of recommendations. Finally, an exemplary event log used in the Process Mining book by Wil van der Aalst (van der Aalst 2011) as well as the event log of the 2012 BPI challenge (van Dongen 2012) were used to illustrate the RS's ability to handle PFAs of considerable size as well.

5.1 Correctness

Tables in this section will consist of the following columns:

AR: Amount of recommendations

WF: Weight factor

SPE: Series of past events

Expected SPE based: Expectation for recommendations based on SPE. May be empty if SPE ends with a transition that's not part of any PFA in the database.

Expected additional: Expectation for recommendations based on frequency of occurrence of all transitions over all PFAs in the database. May be empty if at least *AR* recommendations can be made from SPE

Actual SPE based: Actual output from the Recommender based on *SPE* including the probability of each recommendation.

Actual additional: Actual output from the Recommender based on transition frequencies including the probability of each recommendation

Starting with the most basic case, i.e. a database consisting of only PFA_1 from Figure 5, the tests depicted in Table 1 were performed. As PFA_1 constitutes a rather trivial testcase, it is easy to figure out the expected results by hand to verify the results of the mathematical operations required to reach the recommendations. Also, first indicators that **R3** and **R4** are fulfilled are visible in row 3 and 4.

Moving on to slightly more challenging testcases, we add PFA_3 illustrated in Figure 7 to the database. Table 2 shows the results of testcases involving both PFA_1 and PFA_3. Row 1 provides first evidence for the fulfillment of **R6** while rows 2 and 3 showcase that **R5** is fulfilled as the additional recommendations always include *(A)* and *(B)* which each have a frequency of 2 over both PFAs compared

AR	WF	SPE	Expected SPE based	Expected additional	Actual SPE based	Actual additional
1	2	A	B	-	B (1,0)	-
2	2	A	B	A	B (0,73)	A (0,27)
3	2	A, B	EOP	A B	EOP (0,87)	A (0,06) B (0,06)
1	2	-	A	-	A (1.0)	-

Table 1 - Testcases based on PFA$_1$. Source: Self-made

AR	WF	SPE	Expected SPE based	Expected additional	Actual SPE based	Actual additional
2	2	B	E, EOP,	-	EOP (0,62) E (0,38)	-
3	2	C	D	A B	D (0,57)	B (0,21) A (0,21)
3	2	F	-	A B ?	-	A (0,42) B (0,42) E (0,16)
5	2	F	-	A B C D E	-	A (0,32) B (0,32) D (0,12) E (0,12) C (0,12)

Table 2 - Testcases based on PFA1 and PFA3. Source: Self-made

to all other transitions with respective frequencies of 1. Note that in row 3, there was no clear expectation for the third recommended additional transition due to

that fact. Instead of *(E)*, both *(C)* and *(D)* were equally as likely to be recommended as further indicated by row 4.

Requirements **R1** and **R2** can be tested using PFA_4 as illustrated in the upper half of Figure 16. Rows 2 and 3 of Table 3 show that an *SPE* of increasing size but ending with the same element increases the confidence of the RS in its recommendation, substantially increasing the probability of *(D)* while simultaneously lowering the probability of *(B)*, thereby fulfilling **R1**. Row 4 then illustrates that changing the last element of *SPE* to one that does not occur in the PFA, namely *(F)*, reduces the confidence to zero, only yielding additional recommendations instead of *SPE* based recommendations.

Row 1 further supports the fulfillment of **R6**, recommending *(B)* with more than twice the probability as *(D)* due to state 2 having a higher overall state probability than state 6 in PFA_4. However, it also indicates a weakness of the RS: Due to guaranteeing foremost that longer subsequences of *SPE* lead to higher predictions, the importance of state probabilities might be undervalued. Considering state 2 has probability 0,9 while state 6 has probability 0,1, the actual recommendations of 0,69 for *(B)* and 0,31 for *(D)* do not accurately reflect the likelihood of each transition. Since the ranking of recommendations is still reflected correctly, this drawback is tolerable since it in turn leads to an adequate representation of differences in *SPE* length when making recommendations as displayed in rows 2 and 3.

Row 5 and 6 of Table 3 showcase how the confidence of the RS in its *SPE* based predictions grows with the length of *SPE*. In row 5, *SPE* has length 1. Therefore, the probability to land on a state of which *(A)* originates is contributes significantly to the recommendation, leading to a rather confident prediction of *(B)*, but a cautious prediction of *(D)*, only slightly surpassing the additional prediction *(A)* which is recommended due to *(A)* appearing twice in the PFA. However, once we add *(C)* to *SPE* in row 6, the confidence in predicting *(D)* grows immensely, surpassing even *(B)* now and showing a clear difference between *SPE* based predictions and additional predictions.

At last, testcases containing PFA_1, PFA_3, PFA_4 and PFA_5 were run. The result of those are displayed in Table 4 and illustrate the ability of the RS to handle multiple

AR	WF	SPE	Expected SPE based	Expected additional	Actual SPE based	Actual additional
2	2	A	B D	-	B (0,69) D (0,31)	-
2	2	C, A	B D	-	D (0,57) B (0,43)	-
2	2	B, C, A	B D	-	D (0,79) B (0,21)	-
2	2	B, C, F	-	A B	-	A (0,5) B (0,5)
3	2	A	B D	A	B (0,54) D (0,24)	A (0,22)
3	2	C, A	B D	A	D (0,49) B (0,37)	A (0,15)

Table 3 - Testcases based on PFA4. Source: Self-made

PFA as well as circles within a PFA. While rows 1 and 2 do not offer considerable additional insight compared to the last sets of testcases, row 3 and 4 showcase special requests for the RS, namely requesting the likelihood of all transitions following *(A)* as well as the likelihood of all transition to start a process. Defining *AR* with a value higher than the number of unique transitions within all PFAs returns recommendations for all possible transitions, with the added information whether the recommendation is based on *SPE* or just an estimation based on the frequency of the transition itself.

Additional testcases for chapter 5 of the Process Mining book (van der Aalst 2011) as well as the 2012 BPI challenge (van Dongen 2012) can be found in the appendix. Due to their complexity. the actual PFAs were not included and results can therefore not be evaluated manually by the reader.

AR	WF	SPE	Expected SPE based	Expected additional	Actual SPE based	Actual additional
3	2	C, A	B C D	-	D (0,45) B (0,34) C (0,21)	-
3	2	E, D	B E EOP	-	B (0,36) E (0,36) EOP (0,28)	-
99	2	A	B C D	A E	B (0,36) D (0,26) C (0,22)	A (0,10) E (0,06)
99	2	-	A B C	D E	A (0,51) C (0,20) B (0,13)	D (0,08) E (0,06)

Table 4 - Testcases based on PFA1, PFA3, PFA4 and PFA5. Source: Self-made

5.2 Runtime

Separate tests must be performed for the Reader and the Recommender. Additionally, since the Reader can load the database once and then read multiple tsml files at a time, but can also be used for single tsml files, a runtime test must include both options. The Reader's runtime scales with the size of input PFA, i.e. their number of states and transitions, as well as with the size of the database. The Recommender's runtime scales with the length of *SPE*, the amount of recommendations and the size of the database.

5.2.1 Reader

Testing the Reader first, Table 5 displays the following parameters, starting with an empty database in every single testcase:

Amount: The amount of times the input PFA is read by the reader

Amount	States	Transitions	At once (ms)	Separately mean (ms)	Separately total (s)
1000	3	2	649	518	518
500	6	6	580	388	194
1000	6	6	1216	885	885
200	15	32	675	504	1001
100	52	211	1524	1374	137
5000	52	211	62558	???	???

Table 5 - Runtime evaluation of the Reader. Source: Self-made

States: The number of states in the input PFA

Transitions: The number of transitions in the input PFA

At once (ms): Time in milliseconds for the Reader to terminate if the database is loaded once and all PFAs are read afterwards

Separately mean (ms): Mean time in milliseconds it takes for the Reader to load the database and read a single PFA when reloading the database each time

Separately total (s): Total time in seconds it takes for the Reader to load the database and read all PFAs when reloading the database each time

Examining the results of Table 5, it is easy to see that loading and storing the database constitutes the bottleneck of the Reader. The average time needed to read a single PFA is only marginally lower than the time it takes to read several PFAs without saving and reloading the database after each iteration. Reading bigger PFAs as expected takes longer time than reading smaller PFA, but even 5000 iterations of the 2012 BPI challenge can be read in close to a minute. For comparison, the creation of the respective PFA from the 70 MB event log file took around 30 hours on the same machine.

Finally, loading PFAs separately into the database would require a considerable amount of time from the user himself in practice. Since all PFAs within the data folder are read during a single execution of the Reader, the user would have to move tsml files into the data folder one at a time and start the reader after each step which, for smaller PFA, would take longer than the Reader itself takes to

execute. Therefore, the runtime tests in column 4 reflect the reality much better than those in column 5 and 6 and show that the Reader is efficient at handling large amounts of big PFAs as showcased in row 6, where 5.000 iterations of the 2012 BPI challenge were read in just a minute. Reading the PFA separately was not tested in this case due to the expected runtime being excessively high.

5.2.2 Recommender

As mentioned previously, the runtime of the Recommender depends on the length of *SPE*, the amount of recommendations and the size of the database. Since each PFA is treated separately, having multiple copies of the same PFA in the database does not diminish the runtime of the Recommender. For ease of use, several copies of the same PFA were used in the testcases outlined in Table 6. It contains the following columns:

Amount: The amount of PFAs stored in the database.

States: The amount of states of a single PFA in the database

Transitions: The amount of transitions of a single PFA in the database

AR: The user specified amount of recommendations

SPE length: The length of *SPE* as specified by the user

Time (ms): The time in milliseconds it takes for the Recommender to execute.

The results indicate that the runtime of the Recommender is mostly dependent on the size of the database. In fact, for the testcases in rows 6-8, loading the database accounted for roughly 95% of the total runtime of the Recommender. Therefore, the effect of *SPE* length as well as amount of recommendations on the total runtime is only relevant in larger databases, as exemplified by the difference in runtime between row 6 and 7 as well as 6 and 8 respectively.

6 Conclusion

The goal of this thesis is to design and implement a RS for business processes compatible with PFA created by RegPFA. During the design phase, several requirements for the RS were determined and their fulfillment was verified at several points within the implementation and evaluation process. The resulting

Amount	States	Transitions	AR	SPE length	Time (ms)
250	6	6	10	0	388
250	6	6	10	2	363
250	6	6	1	2	361
5000	6	6	1	2	4615
5000	6	6	1	0	4620
2500	52	211	10	0	41288
2500	52	211	10	2	50933
2500	52	211	3	2	42791

Table 6 - Runtime evaluation for the Recommender. Source: Self-made

tool, RegPFA RS, offers a fast and efficient way to create a database containing several thousands of variably sized PFAs and provide recommendations based on that database within minutes.

The RS can be customized by users through changing several parameters, namely *SPE*, the already transpired events that a recommendation is based on, *AR*, the amount of recommendations, and *WF*, the weight factor used to diminish the relevancy of recommendations based on shorter subsequences of *SPE*. Additionally, a basic fundament for different weighing methods and sort algorithms was built which can be further expanded to provide the user with even more customization options.

The drawbacks of the RegPFA RS are solely related to state probabilities. For large PFA, calculating state probabilities is impossible to do in reasonable time due to the amount of calculation steps scaling not just exponentially, but factorially. While they will be calculated for smaller PFA, their relevance is strictly secondary to the length of *SPE* by design. As explained in chapter 5.1, this leads to a ranking of recommendations in line with the determined requirements while providing possibly misleading probabilities.

It can therefore be concluded that while the goal of designing and implementing a RS for business processes was reached, the actual implementation is lacking in some regards. Since the evaluation of correctness and runtime indicate no

issues though, possible improvements to the behavior of the Recommender regarding state possibilities can easily build on the intermediate results the Recommender provides.

7 Bibliography

Breuker, D., Matzner, M., Delfmann, P., Becker, J. (2016): Comprehensible Predictive Models for Business Processes. In Management Information Systems Quarterly (MISQ) Volume 40(4), pp.1009-1034

Chen, H., Ching, R. H. L., Storey, V. C. (2012): Business Intelligence and Analytics: From Big Data to Big Impact. In Management Information Systems Quarterly (MISQ), Volume 36(4), pp.1165-1188

Covington, P., Adams, J., Sargin, E. (2016): Deep Neural Networks for YouTube Recommendations. In RecSys '16 Proceedings of the 10th ACM Conference on Recommender Systems, pp.191-198.

Felfernig, A., Friedrich, G., Schmidt-Thieme, L. (2007): Introduction to the IEEE Intelligent Systems Special Issue: Recommender Systems. In IEEE intelligent systems, Volume 22(3), pp.18-21. DOI: 10.1109/MIS.2007.52

Gomez-Uribe, C. A., Hunt, N. (2016): The Netflix Recommender System: Algorithms, Business Value and Innovations. In ACM Transactions on Management Information Systems (TMIS), Volume 6(4), Article 13.

Isinkaye, F. O., Folajimi, Y. O., Ojokoh, B.A. (2015): Recommendation systems: Principles, methods and evaluation. In Egyptian Informatics Journal, Volume 16(3), pp.261-273

Portugal, I., Alencar, P., Cowan, D., (2015). Requirements Engineering for General Recommender Systems. In arXiv:1511.05262

Shani, G, Heckerman, D, Brafman, R. (2005): An MDP-Based Recommender System. In Journal of Machine Learning Research 6, pp.1265-1295.

Smith, B., Linden, G. (2017): Two Decades of Recommender Systems at Amazon.com. In IEEE Internet Computing, Volume 21(3), pp.12-18. DOI: 10.1109/MIC.2017.72

Tondji, L. N. (2018): Web Recommender System for Job Seeking and Recruiting (Unpublished master's thesis). African Institute for Mathematical Sciences, Senegal

van der Aalst, W. M. P. (2011): Process Mining: Discovery, Conformance and Enhancement of Business Processes. Retrieved from http://www.processmining.org/book/start

van der Aalst, W. M. P., Rubin, V., Verbeek, H. M. W., van Dongen, B. F., Kindler, E., Günther, C. W. (2010): Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting. In Software and Systems Modeling, Volume 9(1), pp.87-111

Van Dongen, B.F. (2012): BPI Challenge 2012 – Event Log of a Loan Application Process. Eindhoven University of Technology

Vera-Baquero, A., Colomo-Palacios, R., Molloy, O. (2013): Business Process Analytics Using a Big Data Approach. In IT Professional, Volume 15(6), pp.29-35

Vidal, E., Thollard, F., De La Higuera, C., Casacuberta, F., Carrasco, R. (2005): Probabilistic Finite-State Machines – Part I. In IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 27(7), pp. 1013-1025. DOI: 10.1109/TPAMI.2005.147

8 Appendix

8.1 Inductive proof of upper limit for merge algorithm

Base step: *TP (1) <* $\lfloor P_1 \rfloor + 1$

$$TP(1) = P_1 < \lfloor P_1 \rfloor + 1$$

This holds by default since by definition of the floor function, $\lfloor x \rfloor > x - 1 \; \forall x \in \mathbb{R}$

Inductive step: *TP(i+1) <* $\lfloor P_1 \rfloor + 1$

$$TP(i+1) = TP(i) + \frac{\left(\lfloor P_1 \rfloor + 1 - TP(i)\right) * P_{i+1}}{(\lfloor P_1 \rfloor + 1) * (\lfloor P_1 \rfloor + 1 - \lfloor P_{i+1} \rfloor)^{WF}} \qquad (1)$$

To simplify the equation, we will first assume that *WF = 0*. That leads to:

$$TP(i+1) = TP(i) + \frac{\left(\lfloor P_1 \rfloor + 1 - TP(i)\right) * P_{i+1}}{(\lfloor P_1 \rfloor + 1)} \qquad (2)$$

Inductively, we assume that we already proved that *TP(i)* < $\lfloor P_1 \rfloor + 1$ holds true.

This is equivalent to:

$$\exists x \in \mathbb{R}: x > 0 \; \wedge TP(i) + x = \lfloor P_1 \rfloor + 1 \qquad (3)$$

$$\Leftrightarrow \; \exists x \in \mathbb{R}: x > 0 \; \wedge x = \lfloor P_1 \rfloor + 1 - TP(i)$$

Now we can replace $\lfloor P_1 \rfloor + 1 - TP(i)$ in (2):

$$TP(i+1) = TP(i) + \frac{x * P_{i+1}}{(\lfloor P_1 \rfloor + 1)}$$

$$\Leftrightarrow \; TP(i+1) = TP(i) + x * \frac{P_{i+1}}{(\lfloor P_1 \rfloor + 1)} \qquad (4)$$

Remembering we assumed without loss of generality that P_1 is the maximum value in *LR*, we know that

$$P_{i+1} \leq P_1$$

$$\Leftrightarrow P_{i+1} < \lfloor P_1 \rfloor + 1 \quad (5).$$

That means that

$$\frac{P_{i+1}}{\lfloor P_1 \rfloor + 1} < 1$$

$$\Leftrightarrow x * \frac{P_{i+1}}{\lfloor P_1 \rfloor + 1} < x \qquad (6)$$

Which, starting from formula (4), leads us to

$$TP(i+1) = TP(i) + x * \frac{P_{i+1}}{(\lfloor P_1 \rfloor + 1)} < TP(i) + x = \lfloor P_1 \rfloor + 1 \qquad (7)$$

This proves the assumption for *WF = 0*. If we assume $WF \geq 1$ now, keeping in mind that *WF* is of integer value, adjusting formula (4) to include *WF* leads to

$$TP(i+1) = TP(i) + x * \frac{P_{i+1}}{(\lfloor P_1 \rfloor + 1) * (\lfloor P_1 \rfloor + 1 - \lfloor P_{i+1} \rfloor)^{WF}}$$

$$\Leftrightarrow TP(i+1) = TP(i) + x * \frac{P_{i+1}}{(\lfloor P_1 \rfloor + 1)} * \frac{1}{(\lfloor P_1 \rfloor + 1 - \lfloor P_{i+1} \rfloor)^{WF}} \qquad (8)$$

Remembering again that P_1 is the maximum value in *LR*, it holds that

$$P_{i+1} \leq P_1$$

$$\Leftrightarrow \lfloor P_{i+1} \rfloor \leq \lfloor P_1 \rfloor$$

$$\Leftrightarrow \lfloor P_{i+1} \rfloor + 1 \leq \lfloor P_1 \rfloor + 1$$

$$\Leftrightarrow 1 \leq \lfloor P_1 \rfloor + 1 - \lfloor P_{i+1} \rfloor$$

$$\Leftrightarrow 1^{WF} \leq (\lfloor P_1 \rfloor + 1 - \lfloor P_{i+1} \rfloor)^{WF}$$

$$\Leftrightarrow \frac{1}{1^{WF}} \geq \frac{1}{(\lfloor P_1 \rfloor + 1 - \lfloor P_{i+1} \rfloor)^{WF}}$$

$$\Leftrightarrow \frac{1}{(\lfloor P_1 \rfloor + 1 - \lfloor P_{i+1} \rfloor)^{WF}} \leq 1 \qquad (9)$$

Applying formula (9) to formula (8) leads to

$$TP(i+1) = TP(i) + x * \frac{P_{i+1}}{(\lfloor P_1 \rfloor + 1)} * \frac{1}{(\lfloor P_1 \rfloor + 1 - \lfloor P_{i+1} \rfloor)^{WF}}$$

$$\leq TP(i) + x * \frac{P_{i+1}}{(\lfloor P_1 \rfloor + 1)} * 1$$

$$= TP(i) + x * \frac{P_{i+1}}{(\lfloor P_1 \rfloor + 1)}$$

which was already proven to be smaller than $\lfloor P_1 \rfloor + 1$ in formula (7).

We have thus shown that *TP (1)* < $\lfloor P_1 \rfloor + 1$ and

$\forall i \in \mathbb{N}: TP(i) < \lfloor P_1 \rfloor + 1 \Rightarrow TP(i+1) < \lfloor P_1 \rfloor + 1$

8.2 Additional PFA

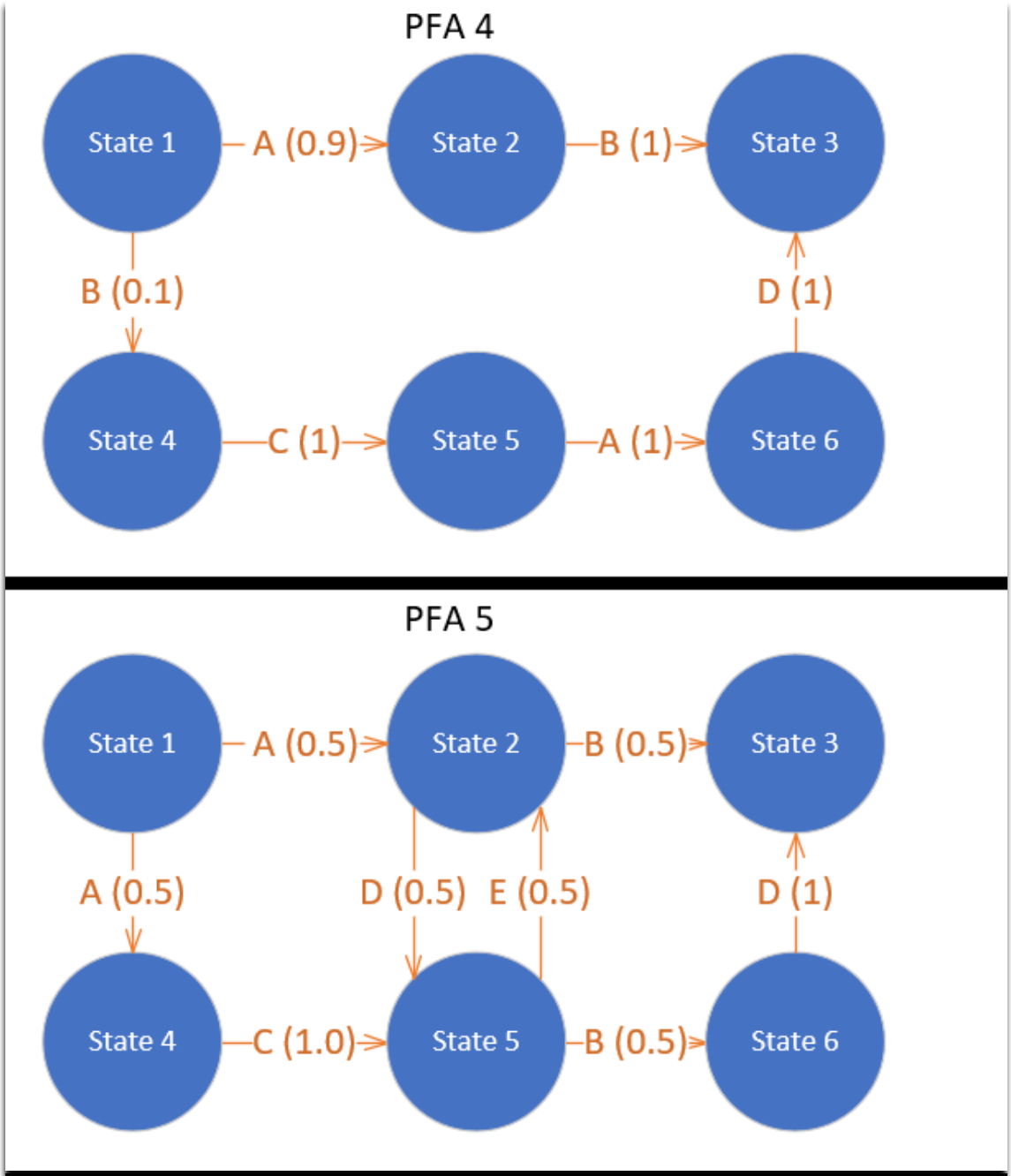

Figure 16 – PFA$_4$ and PFA$_5$. Source: Self-made

8.3 Additional testcases

As mentioned in chapter 5.1, the complexity of the PFAs used to run the testcases in Table 7 and Table 8 impedes their display within this thesis. However, the results were evaluated by hand and found to be accurate. They can also be replicated by creating the respective PFA with RegPFA using the following parameters:

AR	WF	SPE	Actual SPE based	Actual additional
3	2	-	A (0,66)	D (0,17) C (0,16)
5	2	A, C, D	E (0,79) B (0,08) C (0,08)	D (0,03) H (0,02)
5	2	F, D	B (0,45) C (0,27) E (0,19)	D (0,05) H (0,04)

Table 7 - Testcases based on Chapter 5 of the Process Mining Book. Source: Self-made

bigger_example.mxml of Chapter 5 of the Process Mining book (van der Aalst 2011) as shown in Table 7:

Minimum states: 4

Maximum states: 24

EM iterations: 100

Threshhold EM algorithm: 0,001

Pruning ratio: 1,5

BPI_Challenge_2012.xes of 2012 BPI Challenge (van Dongen 2012) as shown in Table 8:

Minimum states: 25

Maximum states: 100

EM iterations: 100

Threshhold EM algorithm: 0,05

Pruning ratio: 1,0

AR	WF	SPE	Actual SPE based	Actual additional
5	2	-	a_submitted(complete) (0,48)	w_nabellen_offertes(complete) (0,13)
				a_declined(complete) (0,13)
				a_cancelled(complete) (0,13)
				w_cmpleteren_aanvraag(complete) (0,13)
5	2	a_submitted(complete)	a_partlysubmitted(complete) (0,47)	w_nabellen_offertes(complete) (0,13)
				a_declined(complete) (0,13)
				a_cancelled(complete) (0,13)
				w_cmpleteren_aanvraag(complete) (0,13)
5	2	a_partlysubmitted(complete)	a_declined(complete) (0,22)	w_nabellen_offertes(complete) (0,13)
			w_afhandelen_leads(schedule) (0,22)	
			a_preaccepted(complete) (0,22)	
			w_beoordelen_fraude(schedule) (0,22)	
10	2	w_nabellen_offertes(complete)	a_declined(complete) (0,16)	w_nabellen_offertes(complete) (0,01)
		w_completeren_aanvraag(start)	w_completeren_aanvraag(complete) (0,16)	o_cancelled(complete) (0,01)
			a_cancelled(complete) (0,16)	w_completeren_aanvraag(start) (0,01)
			a_accepted(complete) (0,15)	
			o_selected(complete) (0,15)	
			a_finalized(complete) (0,12)	
			w_beoordelen_fraude(schedule) (0,07)	

Table 8 - Testcases based on the 2012 BPI Challenge. Source: Self-made