Eindhoven University of Technology

MASTER

Constraint based process monitoring

Selman, J.W.M.

Award date:
2016

Link to publication

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 19. Apr. 2025

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Architecture of Information Systems Research Group

Constraint Based Process Monitoring

Master Thesis

Jasper Selman

Supervisors:
dr. ir. Boudewijn van Dongen
ir. Hans Poppelaars
dr. ir. Huub van de Wetering

Final version

Eindhoven, September 2016

Abstract

We live in a society where the majority of processes is computer-controlled. In each of these processes every piece of information available is stored and every executed action is monitored. These computer-controlled processes relate to the human need to have control over everything and offer opportunities for process monitoring. Companies are constantly looking for new possibilities to improve their processes to stay ahead of the competition. The problem is, since most processes are automated, the process owners often have no idea how their processes exactly behave and they only have a notion of how the processes should behave.
In this thesis, a process monitoring approach is presented to evaluate processes, based on constraints, which enables the user to gain insights in their process. These constraints are extracted from information sources which do know how the process should behave. All the constraints together, define how the process should be executed. The performance of the process is evaluated based on criteria chosen by the users or process owners. This evaluation is done on the process as a whole and on parts of the process individually. To achieve successful results for each case in the process, the evaluation results are used to suggests how cases, currently in process execution, should be continued. To test the approach, it is integrated with the MagnaView platform and evaluated on two real-life case studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the modern business world, companies rely more and more on information systems to manage work. The amount of data stored is growing at an exponential rate. It becomes so huge that it is not possible to manually analyze the data for process improvements. Therefore, the need for automated systems to analyze and monitor the data is growing. This thesis is the result of the graduation project for the Computer Science & Engineering master program at the Eindhoven University of Technology. The project is carried out both at MagnaView B.V. and at the university at the AIS (Architecture of Information Systems) department.

1.1 Problem description

Process mining and process monitoring are popular techniques to gain insights into processes. Process mining is often used to analyze processes, using historical data. The goal of process mining is to discover bottlenecks and other issues in processes. After the analysis a report is produced with recommendations for process improvement. With this report the analysis is finished. For some projects this is not enough, the process needs to be monitored over time. Process monitoring consists of a set of techniques used to monitor the performance of processes over time. The performance of a process is usually measured by *key performance indicators* or *evaluation criteria*. The average throughput time of the process or the number of instances per day are popular evaluation criteria. The main difference between process mining and process monitoring is that process mining is a one-time in-depth analysis, while process monitoring does analyses on a regular basis, but not as in-depth as process mining. On a long term base process monitoring can be more useful for companies than process mining, because the analyses are done more often.

By monitoring a process and using process mining techniques, companies are able to gain insights into their processes regarding performance and bottlenecks. Companies can choose to improve their processes using these new insights, by solving the bottlenecks. It is not always possible to apply any solution found for the bottlenecks in every sector, because some sectors deal with restrictions. For example, the health care sector. The way a patient moves through the processes is regulated by protocols and these protocols cannot be simply adapted to improve several criteria, without endangering the patient. Process monitoring can however still be useful in this sector. Though there is a strict protocol, the protocol is not always followed. It is interesting to see if these protocol violations lead to different results for the

evaluation criteria. These comparisons can result in insight into the effect of following the protocol, e.g. certain parts of the protocol need to be regulated more strictly and other parts have no influence on the evaluation criterion at all. By doing this, the protocol can be evaluated by choosing criteria which are relevant to the process.

Using a protocol, a model can be constructed of all possible paths through the process, e.g. by violating certain parts of a protocol. Using historical data, it is possible to project the score of a criterion for a case which has already started the process but has not finished yet. By choosing the right evaluation criterion, important information can be *predicted* about the case, for suggesting the best continuation action in the case. That is, the action to continue the case execution, which gives the highest possibility to achieve a positive result for the evaluation criterion. Suggesting continuation actions for open cases can be beneficial to people making the decision of how to best continue the case execution.

1.2 Research objective and questions

From the problem description in Section 1.1 the following research objective can be derived:

To develop an approach which evaluates a protocol, as a whole, or the parts within the protocol individually, on a given criterion and predicts the outcome for this criterion for open cases.

The outcome should provide insight into the protocol which defines the process and the influence of the parts within the protocol on the evaluation criterion. The outcome should both be informative and intuitive for users. If the suggestions are followed strictly, the overall score for the evaluation criterion should improve. From this research objective, the following research questions can be derived:

1. How can a protocol and an evaluation criterion be represented, such that it can be evaluated?
2. How can the influence of a protocol on an evaluation criterion be measured?
3. How can the predicted outcome of an evaluation criterion for open cases be used to suggest the best continuation action during execution?
4. How can the outcome of an approach be visualized, such that the result is intuitive for the user?

1.3 Scope

For this thesis, only processes with a clear and strict protocol are considered. For example, the treatment process for a patient in a hospital or the process of granting a mortgage by a bank. The focus of this thesis is to gain insight in the influence of process steps on an evaluation criterion and not on translating every protocol step into a predicate. In this thesis a general method for translating a protocol to constraints is given, but this method is not guaranteed to be complete. Furthermore, the information to derive the outcome of an evaluation criterion for the process needs to be stored in the information system, thus limiting the possibilities of available criteria.

The project is carried out at MagnaView B.V.[1], a company specialized in creating smart products, a combination of visual analytics and business intelligence in the associated field. The visualizations will be created with the MagnaView software.

1.4 Methodology

To achieve the results and conclusions presented in this thesis, the following steps are taken:

1. Conduct a literature study on process mining and process monitoring.
2. Investigate the current available techniques and their usability for the problem sketched in this thesis.
3. Develop a solution to the problem.
4. Test the solution on a simple non real-life instance of a problem.
5. Integrate the solution as a prototype in the MagnaView software.
6. Evaluate the solution on a real-life instance of the test problem (Case Study 1).
7. Evaluate the solution by means of a case study on a less structured process (Case Study 2).
8. Evaluate the results and provide future work.

1.5 Evaluation

To evaluate the solution presented in this thesis, two case studies are conducted. The first study is conducted at the TU/e. This study is the real-life version of the example instance on which the concept will be tested. In short, the order of courses followed at the TU/e are investigated. More information about the problem can be found in Chapter 7.
The second study is conducted in a hospital in the Netherlands. Due to the enormity of all the protocols in a Dutch hospital, only a subset is used to evaluate the solution. For a more elaborate explanation see Chapter 8.

1.6 Outline

The remainder of this thesis is structured as follows: Chapter 2 provides necessary background information and related work which might be useful for the problem sketched. Chapters 3, 4, and 5 are dedicated to answer the research questions. In Chapter 6 the implementation of the approach and important design decisions are discussed. Chapters 7 and 8 provide evaluation on real-life processes, both on structured and less structured processes. The thesis concludes with Chapter 9, in which the final conclusions and possible future work are provided.

[1] www.magnaview.com

Chapter 2

Related Work

In this chapter the different aspects of the concept sketched in Section 1.2 are researched separately. For each aspect it is first discussed what it needs to accomplish and which related work has already been done. In Section 2.1 the basics of process mining and how this thesis fits in the process mining field are discussed. After that, widely used techniques which might be interesting to use as a basis for the construction of constraints are discussed in Section 2.2 and Section 2.3. Once these techniques are discussed, the best suitable technique for this thesis is chosen in Section 2.4. In Section 2.5 additional information about automata can be found, for those who are not familiar with automata. In Section 2.6 recommendation systems are discussed. Such a system will be used as to recommend continuation actions during execution.

2.1 Process mining

Process mining is the term which is used for many of the techniques which focus on extracting processes from data and/or try to improve these processes [34]. It is a combination of data mining and business intelligence, where data mining is used to mine a process from the data and the business intelligence is used to analyze and improve the process. The interest from the business industry in process mining techniques is vastly growing. Firstly, because of the automation in almost every business, it is much easier to record information about processes precisely and store this data. Second, there is a constant need for process improvement such that companies can still compete with other companies. Process mining is a relatively new method to achieve this improvement.

2.1.1 Event logs

Most modern IT systems leave a footprint during its process execution. These footprints consist of recorded events, which occurred during the execution of the process. When all events are bundled it is called an *event log*.
Most of the systems used in real-life processes are extremely dedicated to the process they are designed for and therefore completely different from systems used for other processes. For event logs to be properly used in process mining, it is important that even though all the systems are different they should record at least the same basic information. That is why a few minimal requirements for event logs have been introduced in [14]. These requirements

are the following:

1. Each event in the log refers to one specific activity in the process. These activities should be uniquely identifiable.
2. Each event belongs to a specific case.
3. Each event refers to one certain point in time and not to a period.
4. Each event has a description of the type of event (i.e. start or completed).
5. All events within a case are ordered on their timestamps.

An example of a proper event log is shown in Table 2.1 [8] which represents a fraction of a credit application process. All the requirements stated above are met and the log also contains extra information regarding to the resource and loan attributes and the lifecycle.

Case	Activity	Resource	Timestamp	Loan	Lifecycle
1	Handle Request	Rory	01-01-1970 09:00:00	750	Start
1	Handle Request	Rory	01-01-1970 09:05:00	750	Complete
1	Simple Check	Amy	01-01-1970 10:00:00	750	Start

Table 2.1: Fraction of an example log of a credit application process

2.1.2 Process mining types

Process mining covers many different techniques, but in general there are three main categories in which the work with regard to process mining can be divided. As described by [10] the categories are the following:

1. *Discovery*: A discovery technique takes an event log and creates a model from it, using only the information available in the event log. The most extensive research is done in this category and that is why many different techniques exist to discover processes as discussed in [2], [3], [4], [5].
2. *Conformance*: An already existing or discovered process model is compared with an event log of the same process. From this comparison it can be checked whether the data stored in the event log reflects the process model to see if the process model is correct. See [6], [7], [8], [9].
3. *Enhancement*: An already existing process model is improved, extended or repaired using data from an event log of the same process. This technique could be used to show bottlenecks or throughput times in the process, using timestamps from the event log. It could also be used to alter/repair certain parts of the process which do not reflect the data in the event log adequately. See [9] and [10].

Often process mining techniques are used to analyze processes. For process mining, an event log consisting of data from a period in the past, is used. On this dataset an in-depth analysis is done and at the end a number of process improvements based on this analysis are suggested. These analyses are usually done once, but it is also possible to do the analyses more often, or

even periodically, to monitor the process. When this is done, it is called *process monitoring*. The goal of process monitoring is to monitor the process to find improvements in the process and check if changes in the process indeed improve the process.

Looking at the scope of this thesis, the conformance category has the most similarities with the problem at hand. Since the conformance category has as purpose to check if the data conforms to the process model, which is similar to checking if the data conforms to a protocol. For conformance checking, certain properties are available in a process (because they are provided at start). It is checked if all cases in the data conform to these properties. This might mean that techniques from the conformance category can be of use in process monitoring.
A difference between monitoring and conformance checking, is in the way how the model is evaluated. Standard metrics are used for conformance checking; *fitness*, *precision* and *generalization* [10]. Fitness measures how well the data fits the process model, precision measures if the model only allows for the behavior seen in the log, while generalization works the other way around. For the approach used in this thesis these metrics are interesting, but do not give the desired insight. Because these metrics only give insight in how well the model reflects the data, but do not give insight in specific properties of the process. In the next two sections, two popular techniques for conformance checking are discussed.

2.2 Alignments

The first technique which helps to gain insight in the process are alignments. Alignments show how well a process model fits the associated data by producing a set of metrics. The metrics can be used to monitor processes, e.g. if a change improves the process, the outcome of the metrics should also improve. Using the alignment technique, it can be checked if the model indeed reflects the data collected. It is also possible to evaluate cases individually instead of the process as a whole. This can be done by checking if the process instance can be executed on the model. Alternatively, you can deduct which actions are missing (or are missing in the model, to execute this case), using the alignment for that particular case. Research about this subject has already been done by [1], [6], [13], [15] & [16].
Based on the information discussed in this section, this technique seems suitable to use for process monitoring, however there are issues. A major issue is the fact that the technique as discussed is only applicable on completed cases. Since the scope of this thesis also includes cases which are not yet finished, this problem needs to be solved if this technique is used.

That is not the only issue when using alignments. To apply alignments, it is important that the process model is correct. Correct means that it exactly reflects the process as it is supposed to be executed. If this is not the case, it is possible that cases are falsely marked as cases where something is going wrong, while in reality the model is incorrect. Creating a correct model is not a trivial problem at all, the whole *process discovery* field is devoted to it. Another issue is that the result is complex for people who do not have a process mining or technical background. Every user should have knowledge of alignments and the metrics previously discussed. Even if the user has the knowledge it is still not easy to see what went wrong in a trace and why it went wrong.
Monitoring is used on a more frequent basis than a process mining analysis, which is a one-

time job. It is performed by other users, probably a manager instead of an analyst doing the analysis. Therefore, it is important that the result is still comprehensible and usable. Finally, the process of calculating the alignments is quite expensive in computational cost. In practice this means, that this will be quite slow. Research to improve the performance is still being done [12], but it is still a weak point of the technique. Taking all these issues into account, alignments is not the best option to use for monitoring processes.

2.3 Logics

A different method which can be used to monitor processes is to define a process based on properties of a process. The properties are to be provided by an *information source* with extensive knowledge of how the process should be, such that the constraints reflect what should happen during a process. This information source could be everything, from a person to a book or an established protocol.
An advantage of this technique is that it does not depend on a process model. The model which is used consists of the provided protocol. For this technique, it is necessary to formalize the protocol and the evaluation criteria. A possibility is to represent the protocol using logic. Since the traces, which will be evaluated, have timestamps and are therefore ordered, temporal logics are a suitable choice. Computational Tree Logic, referred to as CTL, and Linear Temporal Logic [20], referred to as LTL, are candidates. Traces on which the criteria are evaluated are all linear in execution, therefore LTL is the best candidate.
In [26] the following is stated: "*Linear Temporal Logic is a widely used logic for expressing properties of programs viewed as a set of expressions*". This is highly applicable to processes, since a process can be seen as a program. Just as for programs, it is possible to define how a process should be executed, or what should not happen. The expressing properties are called *constraints*. Each constraint expresses one property to which an execution set of the program should adhere. An execution set of a program can be seen as a trace, which is an execution set of events defining the process. The idea to use LTL to create constraints for process verification is highly applicable and therefore not new. In [17], [21] and [24] various methods have been designed to apply these constraints for verification of processes. To understand how this logic can be applied to monitor processes, an understanding of Linear Temporal Logic is necessary.

2.3.1 Linear temporal logic

It is not only important to know the language specifics, but it is also important to know what kind of constraints could be expressed using LTL. This has a simple answer; for every constraint, an expression can be made using LTL. The crux of the matter is that a specific set of expressions is very easy and intuitive to create and others are not. In LTL there are four standard operators. With these four operators and basic propositional logic it is possible to create all expressions. These basic operators, where ϕ and ψ are constraints, are the following:

- *Next*(ϕ): in the next event of the trace, ϕ needs to hold. *Next(ϕ)* can also be written as $\bigcirc\ \phi$. For example, take trace *abd* and activity a is just executed. If ϕ is *Activity* = *d*, then $\bigcirc\ \phi$ evaluates to false, since for the second event *Activity* = *b*, but $\bigcirc\ (\bigcirc\ \phi)$ evaluates to true.
- *Finally*(ϕ): somewhere during the execution of the trace, ϕ becomes true. *Finally(ϕ)* can also be written as $\Diamond\ \phi$. For example take *Activity* = *a* as ϕ. This means somewhere in the execution of the trace; activity a is executed.
- *Always*(ϕ): constraint ϕ should hold during each event of the trace. *Always(ϕ)* can also be written as $\Box\ \phi$. An example is that if the current activity is a, it should be directly followed by a b. In LTL this would result in $\Box$ *((Activity = a)* $\rightarrow \bigcirc$ *(Activity = b))*
- *Until*(ϕ, ψ): until ψ holds in the trace, ϕ should hold. *Until(ϕ, ψ)* can also be written as $\phi\ \mathbf{U}\ \psi$. A good example for this is that until activity a is executed, b should never be executed. ϕ is in this case $\neg$*(Activity = b)* and ψ is *Activity = a.*

With these operators specific types of constraints can be expressed relatively easy. Take as an example the constraint that activity a should occur before activity b, or if activity a occurs, b should never occur. Other constraints are more difficult. For example, activity a is the third activity. This would result in in $\bigcirc$ *(*$\bigcirc$ *(*$\bigcirc$ *(Activity = a)))*, which is not a convenient way to express it. More general, with LTL it is straightforward to define the order of activities. It is also possible to refer to data attributes in constraints. *Activity* as used in the expressions above is such a data attribute, but *User* and *Resource* are other examples. It is harder to express counting constraints, like the third activity or the second time the activity occurs.
In [18], [19] a template with numerous expressions, based on LTL, is created which could be used for checking certain properties during execution of a process. However, all this research is done for closed cases and the scope of this thesis also concerns open cases. This is more difficult, since in an open case it is not always possible to predict whether a constraint will be true or false once the case is completed.
An advantage of using LTL formulas is that all these constraints can be easily transformed to automata, which are very powerful aid to solve the open case problem. Each of the constraints can be transformed to an automaton where it can be checked for each state whether the constraint holds or not, but there are modifications which have to be made. This is due to the fact that for regular LTL the expressions are based on infinite executions. In process mining it is never the case that a trace of a process is infinite. More information on how to apply automata can be found in Section 2.5.
The idea to use automata and LTL expressions has been implemented in the Declare framework [30]. The idea behind this framework is similar to the objective of this thesis, though the framework is more suited for loosely structured processes. In this framework constraints can be created using Finite LTL, a variation on LTL. The constraints can be evaluated on traces. Since a variation of LTL is used in Declare, this supports the fact that LTL is a suitable candidate to create the constraints in this thesis. In the next sections, variations of LTL are discussed, to see which variation fits the problem at hand the best.

2.3.1.1 Finite linear temporal logic

A first variation is Finite LTL, or FLTL, introduced by [27]. The difference between FLTL and basic LTL is in the *Next* operator. In basic LTL, the $\bigcirc$ operator always works, since

in infinite traces there is always a next event executed. This is not the case on finite traces. Therefore, a different version of the $\bigcirc$ operator is introduced, the weak $\bigcirc$ operator, denoted as $\oplus$. The $\bigcirc$ operator states that there must be a next state and that in that state ϕ must hold in contrast to the $\oplus$ operator. This operator states that *if* there is a next state, then ϕ must hold. Though this variation already deals with the finite problem, it still has no solution for open cases and empty traces.

2.3.1.2 LTL $^{\mp}$

A second variation consists of two logics, LTL^{+} and LTL^{-}, when taken together referred to as $\text{LTL}^{\mp}$. This variation is introduced by [28] and is able to cope with empty traces. $\text{LTL}^{\mp}$ deals with this in the following way: for LTL^{-} every constraint is satisfied by an empty trace, while the opposite holds for LTL^{+}. Now the question is, when does $\text{LTL}^{\mp}$ follow the semantics of LTL^{-} and when of LTL^{+}? The problem is that the semantics are quite complicated and open cases can still not be evaluated.

2.3.1.3 LTL_3

A variation that can cope with the open case problem is LTL_3 as described in [29]. A third outcome is added for each formula, hence the 3 in LTL_3. This third value is the *undefined* value. The idea of this logic is that for a specific, not closed, trace all the possible paths from the current state to one of the end states are considered. If all these paths result in the same value for constraint ϕ, then the outcome of this constraint is evaluated to the same value. However, if different paths result in different outcomes for ϕ then the outcome becomes *undefined*.

2.3.1.4 RV-LTL & RV-FLTL

Runtime verification LTL, for short RV-LTL, is another variation of LTL and is discussed in [23]. RV-LTL is a combination of LTL_3 and FLTL and works as follows. When LTL_3 evaluates to true ($\top$) or false ($\bot$), this value is also used by RV-LTL. The difference is in the *undefined* case. In this case, RV-LTL resorts to FLTL. If FLTL produces $\top$ for the constraint, the outcome becomes probably true (denoted as $\top^p$). If the outcome is $\bot$, RV-LTL produces probably false ($\bot^p$). This method is able to deal with open cases in such a way that, the result is still useful in the cases where it is not yet known what the outcome would be.
RV-LTL however is only applicable on infinite executions of traces. To make the technique more suitable to finite traces, [25] defined a variant, RV-FLTL, which is applicable on finite traces. In this variant the probably true and probably false have a different definition. $\top^p$ is defined as "ϕ is satisfied at the moment, but there is a possible finite continuation of the trace such that eventually ϕ is not satisfied". $\bot^p$ is defined as "ϕ is unsatisfied at the moment, but there is a possible finite continuation of the trace such that eventually ϕ is satisfied". For short, this logic makes it possible to reason over open cases and make predictions about certain constraints.
Regarding the usability issue as mentioned in the alignment section, these constraints are probably easier to comprehend for the users. The constraints can have any name, so it can be a name users understand, like activity a should occur. The logic on which the constraint is based, is of no importance for the user.

2.4 Overview of the techniques discussed

With all information discussed in the previous sections taken into account, one could create an overview of all the techniques and check the performance on important features. This way the best suitable technique can be chosen. The result is shown in Table 2.2. A ✓ means the technique satisfies the feature, a ○ means the technique is able to deal with the feature but it is not suited to do it. A ✗ means the feature cannot be satisfied by the technique. The features enlisted in the table are the following.

1. *Finite Cases*: the technique is able to cope with finite traces.
2. *Open Cases*: the technique should be able to deal with a trace which has not ended yet.
3. *Usable in Practice*: the technique should be adaptable such that it can be integrated or used without too much overhead. For example, the alignments are not easy to calculate and integrating them within the software requires a significant amount of time. Another example is that $\text{LTL}^{\mp}$ has complicated semantics and is therefore hard to implement.
4. *Case Level*: the technique is able to express constraints per case instead of only the process as a whole.
5. *Expresses Property*: the technique should be able to express a property of the process.
6. *Comprehensible*: the technique should be comprehensible for the end users. The end users are the people who do the actual monitoring and they should be able to comprehend the technique without having to study the background of the technique. To comprehend alignments for example the user should know what fitness is, but when using logics, the user should only have to know what a constraint expresses.
7. *Calculation Time*: the technique can be calculated in a reasonable amount of time.

	Finite Cases	Open Cases	Usable in Practice	Case Level	Expresses Property	Comprehensible	Calculation Time
Alignments	✓	✗	✗	○	✗	✗	✗
LTL	✗	✗	✓	✓	✓	✓	✓
FLTL	✓	✗	✓	✓	✓	✓	✓
$\text{LTL}^{\pm}$	✓	✗	✗	✓	✓	✓	✗
LTL_3	✓	✓	○	✓	✓	✓	○
RV-LTL	✗	✓	○	✓	✓	✓	✗
RV-FLTL	✓	✓	○	✓	✓	✓	✗

Table 2.2: Overview of all the techniques discussed

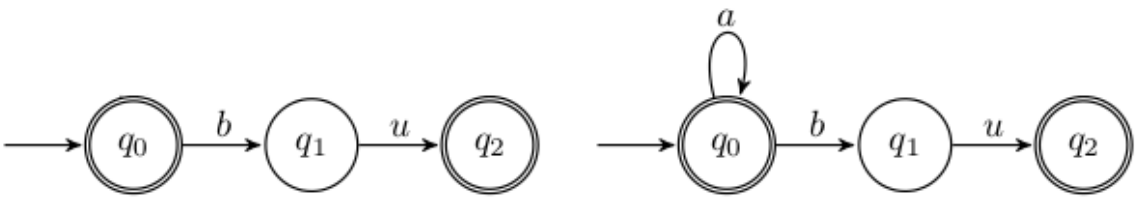
The technique to be chosen should at least adhere to the *Finite Cases* and *Usable in Practice* feature to be considered as basis for this thesis. LTL and RV-LTL already fail for the *Finite Cases* feature and are therefore not useful. By taking the *Usable in Practice* feature in account, alignments and $\text{LTL}^{\mp}$ are also not considered any more. Only FLTL, LTL_3 and RV-FLTL are left as options. For the remaining three techniques, FLTL fails *Open Cases*, LTL_3 fails no features, but does not perform very well for the *Usable in Practice* feature and is also a lot slower to calculate than FLTL. RV-FLTL fails in calculation time, because it is

slow. Therefore, RV-FLTL is not considered anymore.

To eliminate the last technique, it needs to be considered if the failing feature for FLTL is worse than the not so well performing features of LTL_3. Fortunately, for the *Open Cases* feature of FLTL, a workaround can be created. For the open cases, it is enough to know in which states of the automata, generated from each constraint, the case is. This can be done with FLTL. For the not so well performing features, there is no simple workaround and therefore FLTL is chosen as a basis for the construction of the constraints.

2.5 Automata

Automata are a widely used method for defining the state of a process or a part of the process [25], [26] and [22]. According to [33] the definition of a finite automaton is: *"An automaton has a given set of states $\mathcal{S}$, a given alphabet $\mathcal{A}$, a transition relation that explains how to get from one state to the next state, an initial state and a set of final states"*.
This means that a finite automaton can be in different states, depending on which action (from $\mathcal{A}$) is executed. The initial state is the start state of the automaton, which usually is the empty state where no actions are executed yet. The set of final states denotes which language is accepted by the automaton, i.e. the combination of desired series of actions to be executed. A simple example of a finite automaton is shown in Figure 2.1a. The set of states $\mathcal{S}$ is $\{q_0, q_1, q_2\}$, the alphabet $\mathcal{A}$ is $\{b, u\}$. The translation function in this case are the arrows between states with an action above it. The initial state is q_0, denoted with the incoming arrow. The set of final states consists of is q_0 and q_2, denoted with the double edges. For this automaton, the accepting language is a single b followed by a single u or an empty execution set.



(a) Example of a simple finite automaton (b) Example of a simple infinite automaton

Figure 2.1: Simple examples of finite and infinite automata

The automaton as discussed above is a finite automaton. Next to finite automata there are also infinite automata. The difference is that finite automata represent processes which end at a certain point in time, while infinite automata do not. An infinite automaton is shown in Figure 2.1b. This automaton is infinite because of the infinite sequence of a's which can be executed in state q_0.
A real-life example for a finite automaton is a manufacturing process. Once a manufactured good is finished, the automaton is finished. For infinite automata you can think of a software program, which are constantly running and never end.
An automaton is either *deterministic* or *non-deterministic*. When an automaton is determ-

inistic, every combination of actions leads to one single state within the automaton and thus you know exactly in which state the process is. When an automaton is non-deterministic, this is not the case. Actions might lead to multiple states, and you do not know in which state the automaton is. If the b is changed to an a in Figure 2.1b, you have a non-deterministic automaton.

2.6 Recommendation systems for suggestions

An important aspect of this thesis is making suggestions for the best continuation action to continue a case execution. Recommendation systems are not new and research has already been done by [31] and [32]. To make a suggestion for the best continuation action, a recommendation system can be used to classify actions. Many variations for classifications exist in the literature [32]. Some of these variations are based on well-known optimization strategies such as First Come First Served or Earliest Due Date. These strategies often have the goal to minimize the production time or costs for a manufacturing process.
Other systems are based on historical data instead of these existing optimization strategies. These systems also try to optimize an evaluation criterion, such as the throughput time. Different from the systems based on strategies is that possible subsequent action for a case are evaluated using historical data. This historical data consists of an event log with traces which are already finished and also contain the information necessary for the evaluation criterion. In [31] a function is designed which uses weights for the traces in the log. Traces which are relevant for the predicted next action are assigned a higher weight than the less relevant traces. Finally, all the possible actions get a rank and the best continuation action is known. The systems as discussed are just examples of recommendation systems. In theory it is possible to create an infinite number of recommendation systems. The kind of system depends on what the user is trying to achieve with the recommendations. Because of this, our own recommendation system will be designed in Section 4.2.

Chapter 3

Protocol Evaluation

To achieve the goal "*To develop an approach which evaluates a protocol, as a whole, or the parts within the protocol individually, on a given criterion and predicts the outcome for this criterion for open cases*", the research questions associated with this goal have to be answered. In this chapter the questions "*How can the protocol and the given evaluation criterion be represented, such that they can be evaluated?*" and "*How can the influence of the protocol on the evaluation criterion be measured?*" will be answered.
First a running example will be introduced in Section 3.1. This example will be used to clarify the solutions, when necessary. Section 3.2 explains how the input information, the protocol and the evaluation criteria, can be represented in a formal way. In Section 3.3 the representation is applied on the running example. This chapter concludes with Section 3.4, which explains how the influence of the constraints on the evaluation criteria can be measured.

3.1 Running example

In this section a running example is introduced. This example shall be used in Chapters 3, 4 and 5 to help answer the research questions. In this example the order in which courses at the TU/e should be followed is tested, in specific the courses of the Computer Science & Engineering master program. Officially, every course taught at the TU/e has a set of courses which should already be passed before a student starts this course. These courses will be addressed to as prior-knowledge courses.
For example, to take the course Algorithms, one should have passed the course Logics first. If all these prior-knowledge courses are taken into account, an order in which courses should be taken first can be created. One could say a *protocol* for picking courses is formed, which defines the process of the study program. In reality however, following all prior-knowledge courses before the course is started is not mandatory for a student. The result is that students hardly ever take a look at the prior-knowledge courses and just pick courses which seem interesting to them. Even though this order is not taken very strictly, the courses are not marked as prior-knowledge for no reason. The chances of passing the current course are higher, if these prior-knowledge courses are completed.

For this example, only a few courses are taken into account, namely Business Information Systems (BIS), Probability Theory (PT), Advanced Process Mining (APM) and Business Process Simulations (BPS). PT and BIS have no prior-knowledge courses registered, but APM and

BPS do. Before taking APM, BIS has to be completed and before taking BPS, BIS and PT have to be completed.
The head of the department of Computer Science & Engineering is interested if meeting the prior-knowledge courses affects the length of the study for students, more specific, if the chances of graduating in two years (nominally) are affected by these courses. He is also interested in the validity of prior-knowledge courses constraints. For example, is it really necessary to first follow BIS and PT to pass BPS, or is one (or even none) of these courses enough?

3.2 Protocol and criteria formalization

In the business world, processes are constantly evaluated to see whether a slight improvement can be made to optimize the process. In the process mining field this is usually done by analyzing the process, using an event log. The goal of the optimization is to reduce the throughput time or the costs of a case. In this thesis, the evaluation is done in a different manner. Instead of trying to optimize a process, the approach presented tests if the restrictions in a process are necessary or not. The user can choose on which criteria the process needs to be evaluated. Choosing this sort of evaluation, the following input data needs to be provided at start:

- The protocol to be evaluated.
- A set of evaluation criteria.
- An event log containing instances of process executions.

To evaluate a protocol, it is important to define a standard format for the input. This is directly related to the research question *How can the protocol and the given evaluation criterion be represented, such that they can be evaluated?*, which will be answered in this section.

As discussed in Section 2.4, the best suitable technique to represent the constraints from the protocol is FLTL. Choosing this representation ensures that every constraint has an undisputed outcome. The constraints can be evaluated for each case, no matter if it is an open or closed case. Representing each constraint as an FLTL expression means that a translation has to be done from a non-formal form to an FLTL expression. The way the translation is supposed to go shall be discussed by means of the running example in Section 3.3.

The evaluation criteria do not need any formalization. It is only necessary to include the information necessary to calculate the outcome for the criteria in the event log. Though not strictly necessary, the choice is made to put a restriction on the type of evaluation criteria allowed. The only type allowed is a boolean type, which means that every criterion is a yes/no question. There are two reasons to make this restriction.
First, choosing for a yes/no question reduces the evaluation time and makes the evaluation criteria simple to understand for the user. There is a clear distinction in whether the result of the evaluation criterion is desired or not. The user does not need a context to interpret the results of the evaluation to know if it is successful or not. Second, a yes/no question is often simple to answer by evaluating conditions. It is important to realize that the boolean form does not limit the expressive power of the criteria, it only limits the generalization of the criteria. For example, take the often used throughput time as a criterion. With boolean criteria, multiple criteria have to be created (shorter than 1 day, shorter than 2 days, etc.)

instead of the single throughput time criteria. This is not seen as a problem in this thesis, because for a process it is usually not important to know all the values for the throughput time, but only if the throughput time is within a certain threshold, which is a boolean question.
For the event log, no extra standardization is necessary. It is only important that the information to calculate the evaluation criteria is included. The outcome of the criteria either has to be present in the log, or it must be possible to deduct the outcome from other information in the log.

3.3 Formalization of the running example

To see the formalization work in practice, it is applied on the running example in this section. All the information necessary is provided in Section 3.1. The protocol consists of the four courses, BIS, BPS, APM and PT, where BPS and APM have prior-knowledge courses. Extracting the relevant information from the description, this results in the following constraints.

1. The student should complete the course BIS before the course BPS is taken.
2. The student should complete the course BIS before the course APM is taken.
3. The student should complete the course PT before the course BPS is taken.

The constraints need to translated to an FLTL formula. This results in the following formal constraints, which are numbered for ease of future reference. An important remark is that the **U** operator used in the following constraints is a weak variant, which means the right-hand-side does not have to become true during a trace execution.

1. (¬ BPS) **U** BIS
2. (¬ APM) **U** BIS
3. (¬ BPS) **U** PT

The evaluation criterion can also be retrieved from the description, and is as follows.

1. **Pass the Master Computer Science & Engineering in two years.**

This criterion is referred to as *Master in 2 Years* in the remainder of the chapters using this example. This question is in boolean form, since a student either passes the master in two years or he does not.
The event log is shown in Table 3.1 and consists of 1000 traces, represented by students. Each event consists of the completion of a single course. To keep this example simple, a student can only take one course at a time and pick only one of these four courses. Only successful attempts to pass the course are taken into account.
As can be seen in the event log, the information for the evaluation criterion is present in the third column. The fourth column, which represents the constraints each category violates is not present in the event log, but it shown for the user to see whether the trace conforms to the protocol or not.

Trace	Occ.	Pass Ev. Crit.	Constraints Violated
APM - BPS - PT - BIS	65	13	1, 2 & 3
APM - BIS - BPS - PT	65	36	2 & 3
APM - BIS - PT - BPS	65	39	2
APM - PT - BIS - BPS	65	39	2
APM - PT - BPS - BIS	65	15	1 & 2
BIS - BPS - PT - APM	65	59	3
BIS - APM - PT - BPS	80	68	-
BIS - APM - BPS - PT	65	52	3
PT - BIS - APM - BPS	75	71	-
PT - BPS - BIS - APM	65	36	1
PT - BPS - APM - BIS	65	26	1 & 2
PT - APM - BPS - BIS	65	15	1 & 2
PT - APM - BIS - BPS	65	39	2
BPS - PT - BIS - APM	65	39	1 & 3
BPS - PT - APM - BIS	65	16	1, 2 & 3

Table 3.1: Event log of the running example

3.4 Measuring influence of constraints

With the formal constraints and the evaluation criteria known, it is possible to answer the second research question "*How can the influence of the protocol on the evaluation criterion be measured?*". For the influence a measurement called the *success rate* is defined. This rate defines how well a process performs in a certain *category*.
The term category needs an explanation in this context. When evaluating a protocol, the cases can be categorized. It is possible that a particular case only violated a certain part of the protocol. Take for example the trace "PT - BPS - BIS - APM" from Table 3.1. This trace only violates constraint 1, which is BIS $<$ BPS. Based on violations, the cases can be categorized. The different categories shall be explained in Section 5.2, for now it is satisfactory to know what a category is in context of this thesis.
With the term category defined, it is possible to give the definition for the success rate. The success rate for category α is equal to the number of cases in α with a successful evaluation divided by the total number of cases in α. For example take the cases in the event log in Table 3.1. Take the category with the cases that only violate constraint 3. These are the traces "BIS - BPS - PT - APM" and "BIS - APM - BPS - PT". The total number of cases in this category is 130, and the number of successful cases is 111. This gives a success rate of 85,4 %.
Evaluating the process this way, it is possible to measure the influence of a single constraint on the evaluation criteria and compare it to other categories. The user can choose the categories as will be explained in Chapter 5.

Chapter 4

Generating Suggestions

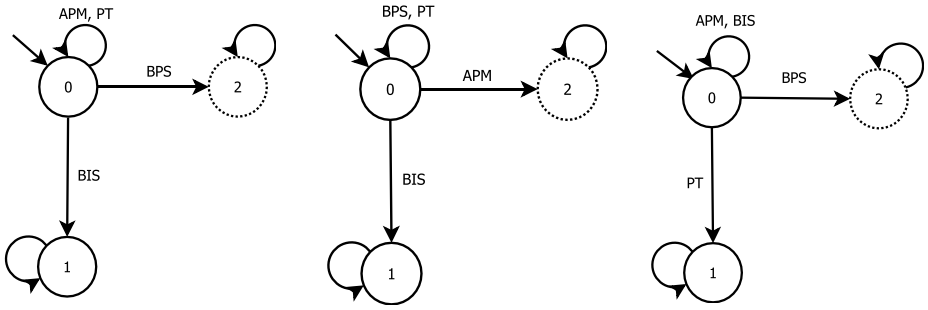
An important part of the research objective is to apply the evaluation of the protocol on open cases. The evaluation provides information about actions during a case execution. The evaluation can therefore be used to suggest continuation actions. The research question concerning this problem is "*How can the predicted outcome of the evaluation criterion for open cases be used to suggest the best continuation action during execution?*" and will be examined in this chapter.
Section 4.1 covers the calculation for predicted outcomes for evaluation criteria. Section 4.2 treats how this predicted outcome can be used to suggest how to continue the execution of a trace. In both sections the discussed information is applied on the running example.

4.1 Predicted values for evaluation criteria

Applications for process monitoring in the business world often offer functionality to monitor cases in their processes. If an application is able to correctly predict which step needs to be taken to continue an open case, it would be of great use for processes. The method designed in this thesis has to offer functionality for this. Therefore the research question "*How can the predicted outcome of the evaluation criterion for open cases be used to suggest the best continuation action during execution?*" is created. To answer this question, it first must be clear what a *predicted outcome of the evaluation criterion for open cases* exactly is.
For each closed case in the process, the outcome of the evaluation criterion is known. For the open cases the outcome is often still unknown. Users benefit if they know if an open case is drifting towards a successful evaluation or not. If a case is likely to evaluate to an unsuccessful result, the user might be able to intervene in time such that the result is still successful. To do this, it is necessary to know what the predicted outcome of the evaluation criterion is for each open case. For the predictions of these outcomes, the evaluation from the previous chapter is used. This outcome, is referred to as the *predicted success rate*. The predicted success rate of the evaluation criterion for an open case is based on the evaluation of all the *similar* cases.
The most straightforward way to define similar cases is "*A closed case is similar to open case α, if for the closed case executed the same actions in the same order as α and did not execute any other action up to the last action of α*". This is a very strict definition. In practice this would limit the cases to compare the open case with drastically, since most processes allow multiple actions at the same time. A small amount of similar cases would make the prediction

unreliable, therefore this definition is not desired.
In the previous chapter, the difference between cases is based on which constraints are violated, not on which actions are executed. It is better to use this difference for the similar cases. The new definition of similar cases is: "*A closed case is similar to open case α if, at some point during execution, the closed case violates exactly the same constraints as α in its current state and does not violate any other constraints*". For example, if there are five constraints and the open case in its current state violates constraint number 2 and 5, all the closed cases which at some point during execution only violated constraint number 2 and 5, are marked as similar cases. It does not matter which actions were performed during the execution to reach this state.

This definition provides a larger group of cases to compare with, but there is still room for improvement. There might be a difference in the non-violated constraints. One case might only need one action to violate a certain constraint and another case needs three actions to violate the same constraint. There is no way to classify the difference, using the current definition. This is where the automata mentioned in Section 2.5 are used. Each constraint in FLTL form can be converted to an automaton. An automaton has the advantage that for each action, it is exactly defined in which state the automaton is. After each action performed during a case, the state of each automaton can change.
Various algorithms exist for the translation step from FLTL formula to automaton. In this thesis an application is used for this translation. Details about this translation can be found in Chapter 6. After the translation, the resulting automata are not necessarily deterministic. It needs to be checked if the automaton is deterministic or not. If it is non-deterministic, the *powerset construction algorithm* [35] can be applied to make it deterministic. The auto-



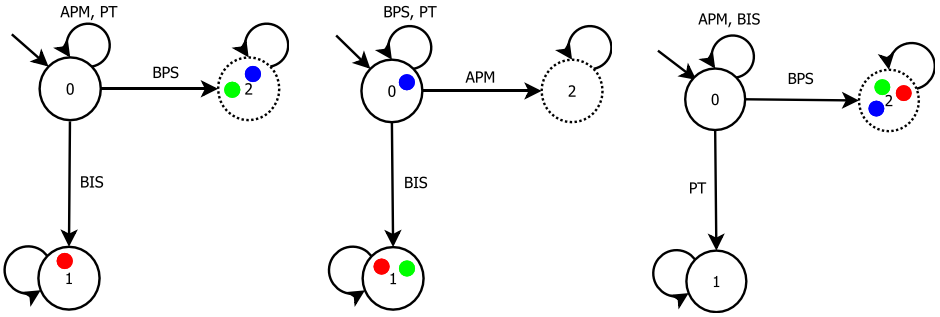
(a) Pass BIS before BPS (b) Pass BIS before APM (c) Pass PT before BPS

Figure 4.1: Automata of the prior-knowledge constraints

mata from the running example are shown in Figure 4.1. These automata are the result of translating the FLTL formulas from Section 3.3. Note that an empty arrow means that every possible action can be done with that arrow. A circle with a normal line means that it is an end state and a circle with a dashed line means that it is not an end state.

It is important that the correct states of automata are compared. Therefore the term *process state* is introduced. A process state is, the set of the states of all the automata which represent a constraint. Compared to the previous definition, a case which is in a process state, needs the same actions to violate a constraint as any other case in the same process state.

The definition which will be used for similar cases is therefore the following: "*A closed case is similar to open case α, if the closed case has, at some point during execution, the same process state as the current process state of open case α*".
Once the similar cases for an open case are known, it is possible to calculate the predicted success rate of the evaluation criterion for an open case. For the predicted success rate, almost the same calculation can be done as for the success rate. A mapping is created where for each closed case, all the different process states in which the case resided is defined. If the current process state of the open case occurs in this list, the closed case is marked as similar. The categorization is now based on similar and non-similar cases instead of violated constraints. For these categories it is possible to calculate the success rate, in this case the predicted success rate.



(a) Pass BIS before BPS (b) Pass BIS before APM (c) Pass PT before BPS

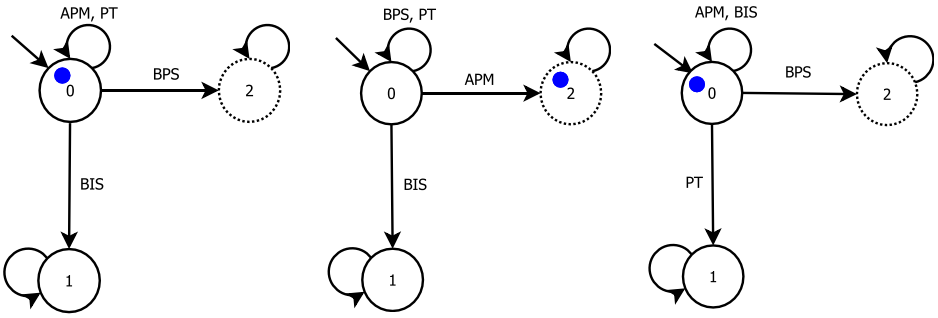
Figure 4.2: Automata with the states of three example cases

The process state of three students in the running example is shown in Figure 4.2. The red student has completed the courses BIS and BPS in that order. The green student has completed the courses BPS and BIS in that order. The blue student has completed the course BPS. The process states of these three students consist of the states of each of the three automata concatenated together. For the red student, the process state is *112*. This means that in the first automaton, the red student is in state 1. In the second automaton, the student is in state 1 and in the last automaton the student is in state 2. For the green and blue student, their associated process states are *212* for the green student and *202* for the blue student.
As an alternative for the process state definition all automata can be merged into one large automaton. One automaton has the advantage that there is only one state at the time for all cases. This allows for simple comparison. However, combining all automata is a very costly operation, while making a set of the states of the automata is not. Another important reason for not choosing this alternative is that it is not straightforward to add or remove constraints. Changing the set of constraints results in a new combined automaton, which is, as mentioned before, a costly operation. Using an automaton for each constraint, this is simpler. Removing a constraint can be done by removing the state from the set of constraints. Adding a constraint results in a new automaton, but a much simpler automaton than the automaton for the combined constraints.
A second alternative is to combine all the FLTL formulas, before creating an automaton. The problem with this alternative is that it does not work for all constraints. For example, it is possible that there are two or more constraints which are mutually exclusive. This means

it is impossible that both constraints are not violated at the same time. When these FLTL formulas are combined, the resulting automaton will be empty and nothing can be compared. This does not happen when each automaton only represents a single constraint.

4.2 Suggestions for open cases

With the predicted success rate defined, it is almost possible to answer the research question "*How can the predicted outcome of the evaluation criterion for open cases be used to suggest the best continuation action during execution?*". It is only necessary to define the *best continuation action*. An open case resides in the process and various actions are possible to continue the execution of the case. For each of the possible actions a predicted success rate is known. The best continuation action would be the action which results in the highest predicted success rate after execution of the action. Note that a high predicted success rate does not ensure a positive result for the evaluation criteria, but the odds are better than for a low predicted success rate.
The predicted success rate has to be calculated for every *relevant* action per case. Relevant actions are actions which cause one or more automata to do a state change. The reason to only calculate the predicted success rate for these actions is simple. If a non-relevant action is performed, no state change occurs and therefor the set of similar cases does not change and the predicted success rate is the same. So in theory at any moment during execution, it should not matter if a non-relevant action is executed.
In the running example, the relevant actions can be deducted from the automata shown in Figure 4.3. For Figure 4.3a these actions are BIS and BPS. For Figure 4.3b the relevant actions are BIS and APM. For Figure 4.3c the relevant actions are PT and BPS. The total set of relevant actions consist of BIS, PT, APM and BPS. This means that in this example there are no irrelevant actions. If for a student it is possible to choose another course, for example Logics, this course is not marked as a relevant action, since it does not cause a state change in any automaton.
A recommendation system, as treated in Section 2.6, is used to generate the suggestions on how to continue case execution. This system ranks the suggestions for each of the relevant actions based on two aspects. The first is the *new* predicted success rate, which is determined as follows. For each relevant action α, the current trace of the open case is appended with α. For this new trace a new process state is derived. This process state is used to generate a predicted success rate after α is executed.
The second aspect used for the ranking, is the number of cases which are marked as similar after each relevant action is executed. It might happen that an action is marked as relevant but in practice is only relevant in a particular phase of the execution of a case. Since this action is relevant, it is appended to the current trace while this action would never be executed in practice at that time. This would result in the fact that for the new process state no, or only a few, similar cases can be found. As mentioned before, comparisons based on a small amount of cases might give distorted results. This behavior is undesirable and therefore the user can specify the minimum number of similar cases a suggestion has to be based on.
The result of these two aspects is that every suggestion is first ranked if the suggestions is based on enough similar cases. In the two resulting parts, the suggestions are ranked on their predicted success rate. The best continuation action for the open case is therefore the suggestion with enough similar cases and the highest predicted success rate.



(a) Pass BIS before BPS (b) Pass BIS before APM (c) Pass PT before BPS

Figure 4.3: Example student in the running example automata

To help understand the suggestions, an example case is shown in Figure 4.3. This student has only completed the course APM. Therefore the current process state of this case is *020*. This means that the automaton of Figure 4.3a is in state 0, the automaton of Figure 4.3b is in state 2 and the automaton of Figure 4.3c is in state 0. The information for the relevant actions is shown in Table 4.1. As can be seen the relevant action BIS is the best option. This suggestion has plenty of similar cases (130) and has the highest predicted success rate. Compared to the other course BPS, this is logical. Choosing BPS would violate two other constraints, while choosing BIS, the constraint complete BIS before BPS is satisfied. Choosing PT does not really affect the predicted success rate of this particular student.

Relevant Action	Current Trace	Current Process State	Current Predicted Success Rate	Similar Cases	New Trace	New Process State	New Predicted Success Rate	Similar Cases
BIS	APM	020	43,7 %	325	APM - BIS	120	57,7 %	130
APM	APM	020	43,7 %	325	APM - APM	020	43,7 %	325
BPS	APM	020	43,7 %	325	APM - BPS	222	22,3 %	130
PT	APM	020	43,7 %	325	APM - PT	021	41,5 %	260

Table 4.1: Suggestions for trace APM

It is important to keep in mind that these actions are only suggestions on how to continue the trace execution. In many processes the decision on how to continue the case execution is influenced by external factors which are not modeled in the process. It is also important to emphasize that for this thesis it is only possible to reason how this evaluation *should* improve the outcome of the predicted results. To know if the outcome is indeed positively influenced, a case study should be performed where one group of test cases is executed in the normal way and the other group should use the improvements from the evaluation. Unfortunately, this is out of scope for a master thesis project and in several processes this cannot be done at all, for example in a hospital.

Chapter 5

Visualization of the Results

An important aspect of the approach presented in this thesis, is visualizing the results. This aspect is associated to the last research question "*How can the outcome of the approach be visualized, such that the result is intuitive for the user?*". In his chapter the visualizations that have been created to show the results will be discussed. The visualizations have been created with the MagnaView tool developed by MagnaView B.V. Section 5.1 offers information about the basic principles of MagnaView visualizations. It is necessary to understand these principles to use the visualizations correctly.
As stated in Chapter 3 and 4, the objective of the method presented in this thesis is twofold. On one hand the protocol of the process is evaluated and on the other hand suggestions are made for open cases. For both aspects a visualization is created. Section 5.2 explains the visualization created to evaluate protocols. Section 5.3 elaborates on the visualization created for the suggestions. In both sections the visualizations are explained on a higher level of detail. For a more detailed description of every element, see Appendix B. Section 5.4 discusses how the visualizations are evaluated to check if they are intuitive.

5.1 MagnaView visualizations

A visualization in MagnaView consists of a *dashboard*. A dashboard is a combination of visualization elements, such that an overview of (a part of) a process is shown. The number of elements can vary from one to as many as you want. In MagnaView the visualization elements are called *dashels*, for example a certain type of a graph or a table. An example dashboard is shown in Figure 5.3. In this figure, a dashboard containing two dashels (one at the top and one at the bottom) is shown. As can be seen is that this dashboard is quite cluttered. The visualizations created in MagnaView are almost always based on the popular minimalistic design of S. Few [11]. The dashboard shown in Figure 5.3 is contradictory to this design, but this is only due to the fact that most of the margins are removed from this dashboard to minimize space on a page in this thesis. The original dashboards will adhere the minimalistic design and are shown in Appendix A.
Another principle that need to be understood, is the filter principle. Every dashboard can have several filters. These filters are located on the top of the dashboard or on the right.

The two relevant filter types are metric selectors and filters for data selections. A metric selector defines which data aspect of the records is shown on the dashboard and groups the data on this aspect. The *Evaluation Criterion* filter in Figure 5.2 is such a filter. The data selection filters are used to select records based on data aspects. Selecting these records, it is possible to look at specific parts of the data. An example for this filter is the *Applied Prior Knowledge Constraints* filter in Figure 5.1. With this knowledge a user should be able to understand the visualizations.

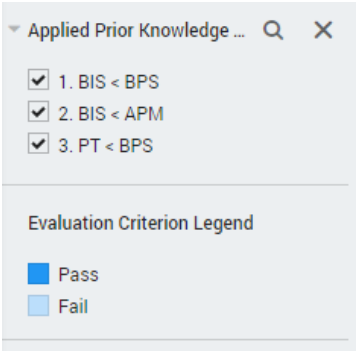


Figure 5.1: Data filter to determine which constraints should be applied

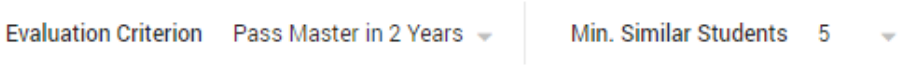


Figure 5.2: The applied evaluation criterion and minimum number of similar cases necessary

5.2 Protocol evaluation dashboard

The dashboard designed for protocol evaluation consists of three dashels, which are shown in Figure 5.3, Figure 5.4 and Figure 5.5. The data in this dashboard is real data of the case study treated in Chapter 7, which is the real-life process of the running example. On each dashel, the categorization is different, to achieve different insights. The first dashel is shown in Figure 5.3. The categorization in this dashel consists of dividing the cases between a violated and a non-violated category. If a case violates any constraint it is placed in the violated category, otherwise in the non-violated category. This dashel is designed to show two things. First, the amount of cases which are executed as *desired* compared to the amount of cases where something went wrong. Second, if violating any constraint has any influence on the evaluation criterion. If this does not happen, the way a case is executed in a process does not matter at all (for the selected evaluation criterion). The second and third dashels are only interesting when the violated category performs worse on the evaluation criteria than the non-violated category.

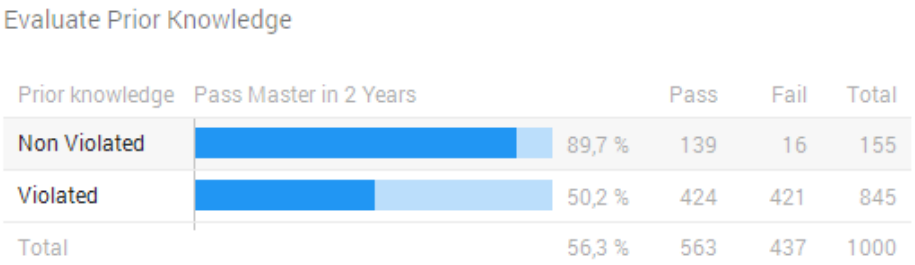


Figure 5.3: The dashel showing the protocol violations

The second dashel is shown in Figure 5.4 and enables the user to investigate smaller parts of the protocol. This dashel categorizes on single constraint violations. When a case violates a

constraint, it is placed in the associated category. If a case does not violate any constraint it is still put in the non-violated category. This means that if a case violates the protocol, it is possible that a case is placed in multiple categories, e.g. the number of categories equals the number of constraints which are violated. Using this level of detail, it is possible to measure the influence of each constraint on their own on the evaluation criterion. This enables the user to investigate which constraints offer room for improvement in the process and which constraints do not.

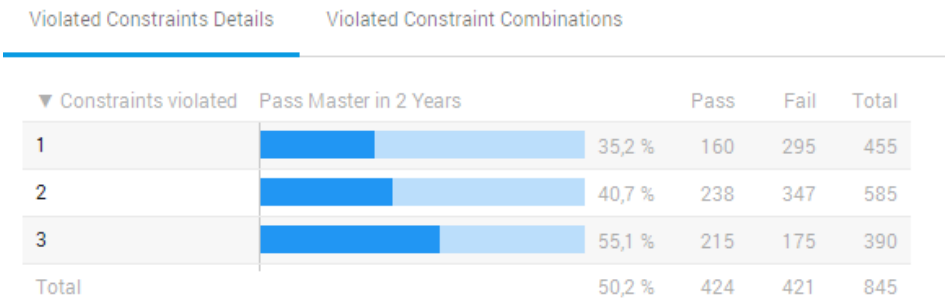


Figure 5.4: The dashel showing the constraint violations

The last dashel categorizes the violated cases in a different way and is shown in Figure 5.4. Instead of a category for each constraint, a category for each combination of violated constraints (encountered in at least one case) is created. With this overview it is possible to analyze possible relation between constraints, i.e. constraint violations always appear together.

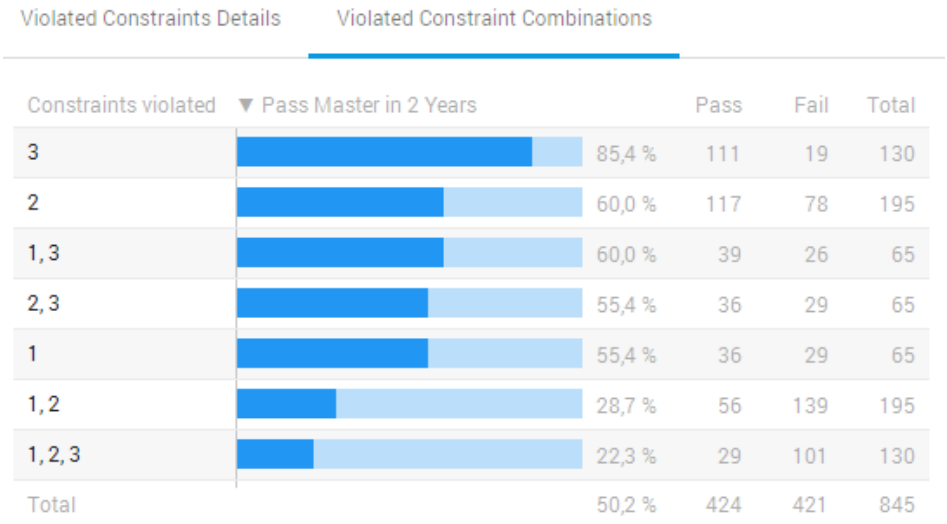


Figure 5.5: The dashel showing the constraint combinations violations

For each of the different dashels, the categories can be compared in the same way, which is based on four statistics. These four statistics are:

- *Success Rate*: the ratio of the number of *Pass* cases in a category divided by all the cases in this category, as defined in Section 3.4.

- *Pass*: the number of cases in a category which have a successful evaluation for the evaluation criterion.
- *Fail*: the number of cases in a category which do not have a successful evaluation for the evaluation criterion.
- *Total*: the number of cases in a category.

The data shown on the three dashels can be manipulated by filters shown in Figure 5.2 and Figure 5.1. The evaluation criterion filter determines the metric on which the process is evaluated. It is possible that a combination of constraints (or one single constraint) is only violated by a couple of cases, therefore the statistics shown can be unreliable. The user is able to determine the amount of cases that are necessary to be reliable and can remove the unreliable results by choosing a value in the minimal similar students filter. With the applied prior-knowledge filter, the user is able to determine which constraints area applied in the process.

5.3 Suggestions dashboard

The dashboard created for the suggestions contains two dashels shown in Figure 5.6 and Figure 5.7. The first dashel contains an overview of all the open cases in the data and is shown in Figure 5.6. Because the amount of open cases can be high, it is possible to make a selection of the visible open cases by using a filter to select visible students. On the other dashel the details for one single case is shown. This case can be selected in the overview dashel by clicking on it.

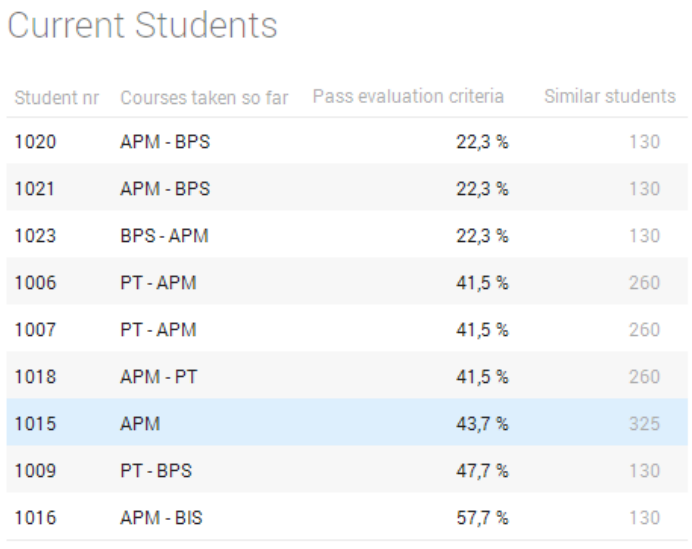


Figure 5.6: The dashel containing the overview of open cases

In the overview of cases, not only the case identifier is shown, but also extra information per case:

- The relevant actions which have already been performed for this case.

- The current predicted success rate for the selected evaluation criterion.
- The number of similar cases on which the predicted success rate is based.
- A warning if not enough similar cases can be found. The user is also able to determine the amount of cases that are necessary to make the given success rate reliable.

The cases in the overview are first sorted on the fact if they have enough similar cases and then on the current predicted success rate. Cases with the lowest success rate are placed on top, since these cases probably need the most attention from the user.
The second dashel, the detailed information dashel, contains additional information per case. In the top of the dashel, the current predicted success rate and the relevant actions performed so far are shown. Below this information, rows are shown, where each row is associated to a relevant action. All the relevant actions are shown as possible continuation actions. For each of these actions, three statistics are presented:

- The new predicted success rate if the associated action is executed.
- The improvement with respect to the current predicted success rate.
- The number of cases on which this new rate is based

It is possible that a problem occurs for a relevant action. There are two types of problems. The first type of problem is that there are not enough similar cases to make a reliable (or even any) prediction for the new success rate. This is shown by a yellow warning icon preceding the name of the action. In certain type of processes, actions are only allowed to be performed once. If processes are evaluated, already performed actions are shown as gray for the predictions, since it is of no use to perform this action again, but it is a relevant action. A real-life example is given by the running example. It is not common to pass a course twice, though it is not forbidden.

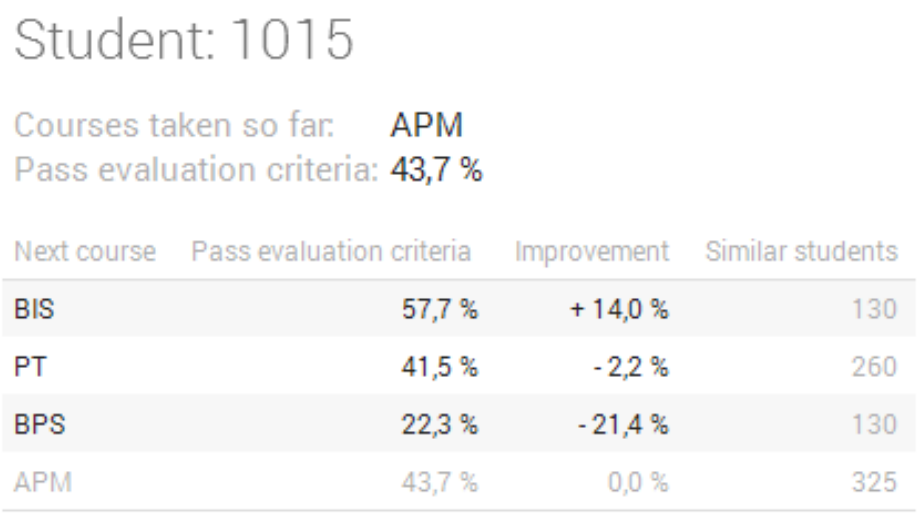


Figure 5.7: The dashel containing the detailed overview of the selected open case

Depending on the evaluation criterion, it is possible that even though the case is not finished, a success or a fail is already achieved. When this happens, the case is grayed out and shown

below all other cases. The success rate is also not shown anymore, but instead depending on the result, a ✓ or ✗ is shown. When this case is selected the detailed information dashel does not show extra information, since no action can alter the result achieved for the evaluation criterion.

5.4 Evaluation process

Though the scope of this thesis is not on the visualization part of the project, it is still desirable to have an intuitive visualization, since this improves the usability. It is customary to execute a formal evaluation for visualization projects, but this is not done for this thesis. The decision to not do this follows from the fact that the visualizations created to make the results of this method more insightful, which is not the main objective of this thesis. Combined with the time consuming factor of the evaluations, those are enough reasons to refrain from doing this. However still several measures are done to evaluate the usability of the visualizations.
The first measure to keep the visualizations intuitive, is that the visualizations are kept close to the style of the default visualizations in MagnaView. One of the benefits of doing this, is that users familiar with MagnaView, already know the interaction between the different dashboard and filters. These users know where to look for the necessary information. Another benefit of the minimalistic design is that there are relatively few elements visible, compared to other dashboards. This means new users are not overwhelmed or even confused, using the visualizations the first time.
To make the visualizations more intuitive, several prototypes of the visualizations are created. These prototypes are shown to several persons, impersonating a user. This group of people both contains people familiar a non-familiar with MagnaView. Their feedback on the visualizations is combined and used to improve the visualizations.

Chapter 6

Implementation

In this chapter the most important implementation decisions are discussed. There are four parts which could be implemented separately.

- The translation from a textual protocol to constraints in FLTL form.
- The translation from FLTL constraints to automata.
- Making sure the automata are all in deterministic form.
- Create a mapping of states in which a case resided during execution.

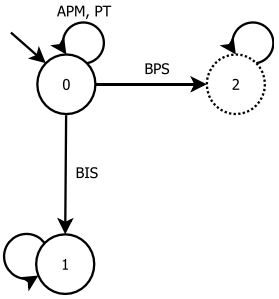
If the translation of a textual protocol to formal constraints has to be efficient, the best option in the scope of this thesis is to do it manually. Since the scope of this thesis is on evaluating processes and not on text interpreters, the choice has been made to manually extract the relevant constraints from the protocol and turn them in FLTL formulas.
The translation of the FLTL formulas into automata is automated, since this work is error-prone if done manually. The choice has been made to use an existing tool to do this and not create one ourself. In existing research, the translation of FLTL formulas to automata has been done often and tools are created for this. The tool used in this thesis is the *Declare* tool, explained in Section 6.1 [1].
Checking if an automaton is deterministic and if not transforming it to a deterministic automaton will also be performed manually. Due to the choice in Chapter 4 to model every constraint as an automaton instead of creating one big automaton for every constraint, the resulting automata are simple. Each automaton only has a few states and transitions. For these automata, the check if they are deterministic can be done manually.
The creation of the mapping of states in which cases resided during execution, will be created with the MagnaView platform. The MagnaView platform is a powerful tool to model data and visualize the results and is therefore suited to create this mapping. The details about this implementation will be discussed in Section 6.2.

6.1 Generating automata with Declare

Declare is a tool developed at the TU/e. It is able to model FLTL formulas in a visual designer. These formulas can be exported in various representations, one of them being an automaton. The formulas are represented by templates, where a template represents a relation between activities, i.e. a constraint.

[1] http://www.win.tue.nl/declare

(a) The constraint pass BIS before BPS modeled in Declare



(b) Pass BIS before BPS

Figure 6.1a shows how a constraint can be modeled in Declare. The blocks represent activities and the lines are relations. The line in this figure represents the constraint that activity *BIS* should be completed before *BPS*. The most common relations are predefined as a template, but if the user wants to model a constraint which is not predefined yet, a template can be created providing the corresponding FLTL formula. The advantage of predefined templates is that for multiple constraints, it is not necessary to first translate them to a FLTL formula. Declare also generates the code such that it can be generated with the Dot framework [36]. This makes it possible to incorporate the automata in other programs. The exported automaton is not guaranteed to be deterministic, so this needs to be checked for each automaton. If it is non-deterministic, it should be made deterministic manually, using the powerset construction algorithm. The automaton resulting from Figure 6.1a is shown in Figure 6.1b.

6.2 Creating evaluations and suggestions with MagnaView

The decision to create the mapping of states in MagnaView, is straightforward. To create the visualizations with the MagnaView platform this mapping has to be present in the input data, so it is convenient to create them with the platform itself. In MagnaView it is possible to create expressions, based on tree logic, to filter the relevant records from the non-relevant and also group the records. These expressions can be added to each record as a new data attribute and it is possible to define the value for this attribute as a user. With this last feature each automaton generated with Declare, can be represented with a single expression. Figure 6.2 shows the expression representing the automaton in Figure 6.1b.

```
if(any(tiles).Trace_subsets = NULL
, 0
    , if((pos("BIS",(any(tiles).Trace_subsets)) = 0) and (pos("BPS",(any(tiles).Trace_subsets)) = 0)//BIS and BPS are both not taken
        , 0
        , if(re_find(any(tiles).Trace_subsets, ".*BIS.*") and not(re_find(any(tiles).Trace_subsets, ".*BPS.*BIS.*"))//BIS is taken bef
            , 1
            , if(re_find(any(tiles).Trace_subsets, ".*BPS.*") and not(re_find(any(tiles).Trace_subsets, ".*BIS.*BPS.*"))//BPS is taken
                , 2
                , if(pos("BIS",(any(tiles).Trace_subsets)) < pos("BPS",(any(tiles).Trace_subsets))//BIS is taken twice, but the first
                    , 1
                    , if(pos("BPS",(any(tiles).Trace_subsets)) < pos("BIS",(any(tiles).Trace_subsets))//BPS is taken twice, but the fi
                        , 2
                        , -1
                    )
                )
            )
        )
    )
)
```

Figure 6.2: The constraint pass BIS before BPS modeled in MagnaView

The set of process states for each case, can be generated from this point. First all the sub-traces have to be derived per case. The number of sub-traces is equal to the number of events per trace. The first sub-trace is the empty trace, the second consists of the first event, the third sub-trace of the first and second event, etc. With the automata expressions and these sub-traces it is possible to determine in which process states a trace has resided during execution. With this mapping, the visualizations proposed in Chapter 5 can be realized.

Chapter 7

Real-Life TU/e Courses Case

In this chapter the same problem as described in the running example is analyzed, only this time using real-life data. More specific, the courses which belong to the Architecture of Information System (AIS) department are evaluated. First the problem shall be described in Section 7.1. In Section 7.2 the problem is formalized such that the concept can be applied. Section 7.3 discusses the quality of the data available for this case study. The results shall be discussed in Section 7.4. In Section 7.5 an intended user is impersonated and an interpretation of the results in context is done.

7.1 Description of the real course system

As mentioned before, this case study tackles the same problem as the problem of the running example. To understand how the course system works at the TU/e, see Section 3.1. For this case study, the head of the AIS department wants to know in which quartile during the academic year a course should be taught. It is important that the distribution of courses does not affect the chance for a student to graduate in two years. To make a decision, all prior-knowledge constraints for courses should be taken into account. The goal of this program is to give the student the best chance to graduate in two years. The current program used is shown in Appendix, C. Nine courses are related to this department. These are the following, enlisted with their abbreviation and study phase:

1. Business Information Systems (BIS), Bachelor
2. DBL Information Systems (DBL), Bachelor
3. Business Process Intelligence (BPI), Bachelor
4. Business Process Management Systems (BPMS), Master
5. Metamodelling and Interoperability (MI), Master
6. Advanced Process Mining (APM), Master
7. Business Process Simulation (BPS), Master
8. Constraint Programming (CP), Master
9. Seminar AIS (SAIS), Master

The courses all connected with arrows in Figure C.1. An arrow from course A to course B means that A is a prior-knowledge course for B. There are two different kind of arrows, the normal arrows are stronger than the dashed arrows. The head of the department believes that students are unlikely to pass courses if they violate the normal incoming arrows for that course. If only dashed arrows are violated, it is harder to pass the course but not unlikely.
The constraints for all these courses are placed in a table to keep it clear which courses are marked as prior-knowledge. Normal arrows are written in bold text. For the remainder of this chapter the normal arrows are referred to as strong constraints while the dashed arrows are referred to as weak constraints.
The head of the department is interested if students graduate in two years if they abide by the constraints stated in Table 7.1. Next to that he also wants to see if there is a difference when a small delay is allowed (about 6 months).

Course	**Prior-Knowledge Course 1**	**Prior-Knowledge Course 2**	**Prior-Knowledge Course 3**	**Prior-Knowledge Course 4**
BIS	-	-	-	-
DBL	BIS *	-	-	-
BPI	BIS *	-	-	-
BPMS	**BIS** [1]	-	-	-
MI	**BIS** [2]	BPMS [4]	-	-
APM	BPMS [5]	BPI *	-	-
BPS	**BIS** [3]	BPMS [6]	-	-
CP	-	-	-	-
SAIS	**BPMS** [10]	MI [7]	APM [8]	BPS [9]

Table 7.1: Prior-knowledge constraints

7.2 Formalization of the problem

An event log of 999 different students is available to use for this evaluation. These students have taken at least one or more courses at the Computer Science & Engineering department. Of these 999 students, 357 are already graduated. The other students are still studying or have stopped studying. Each event in the log is an attempt to pass the exam for a course. For each of these attempts a student can pass the exam or fail it.
By scanning the event log to see which data is available in the log, some of the constraints stated in Table 7.1 can already be removed from the study program. The courses DBL and BPI do not occur in the event log, so the constraints in which these courses appear could not be tested. These constraints are marked with an * in the table. Next to these three constraints, CP is also removed from the protocol since this course does not appear in any constraint. This means no information can be gained about this course. The remaining constraints are numbered like this [1], for future referencing in this chapter.
To keep this case closely related to the example studied in the previous chapters the constraint that PT has to be completed before BPS is added again, since this is a real constraint (constraint 11), but not a course which belongs to the AIS department.
The remaining constraints can all be written in FLTL formulas of the same form as the con-

straints in Section 3.3. This means that there are eleven constraints which can be evaluated. In the translation, no difference is made between the arrow types. The difference should be deductible from the results, since violating the normal arrow constraints should have a stronger impact on the evaluation criteria according to the head of the department.

The evaluation criteria can be extracted from the information stated before, which results in the following two criteria:

1. **Pass the Master Computer Science & Engineering in two years.**
2. **Pass the Master Computer Science & Engineering with at most half a year delay.**

These criteria are referred to as *In Time* and *Delay* in the remainder of this chapter. The necessary information is already available in the event log. Per exam attempt, the date of the attempt, the result and the study phase of the corresponding course are recorded. The duration of the master program can be derived by the difference in the first attempt in the master phase and the last attempt. When several weeks are added to this length, the duration of the master phase is roughly known (because there is some time between the start of a master and the first exam attempt). To get the criteria, this duration can be compared to a constant (two years or two and a half).

7.2.1 Deriving automata

The automata which need to be derived from the FLTL formulas of the previous section are all very similar to each other. Only the labels for each transition are different. The automaton for the first constraint is shown in Figure 7.1. In contradiction to the running example, the failed attempts to pass courses are taken into account. This means that there are several events duplicated, e.g. the event BIS is duplicated to BIS|F (a failed attempt to pass BIS) and BIS|P (a successful attempt to pass BIS). All the courses which are not relevant to the constraints are bundled together in the event *Other Course*.
For this automaton A|$*$ means that the event is either a P or an F for the associated course A. The automaton shown in Figure 7.1 represents the constraint that before a student makes an attempt to pass BPMS, the course BIS should be completed. The other ten automatons are similar to this automaton (only the labels need to be swapped with the correct course) and are therefore not shown.

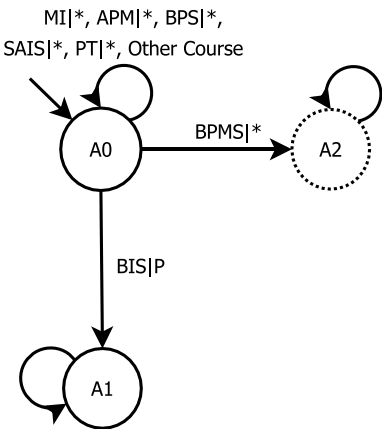


Figure 7.1: Example automaton for constraint 1, BIS|P < BPMS

7.3 Data quality

The dataset used for the evaluation contains a total of 999 students. Of these students 357 are graduated. This is only a small fraction of the total set. In the ideal scenario, the number of graduated and non-graduated students are interchanged. This has the benefit that the categories have more similar cases and the results would therefore be more reliable.

One reason that the fraction of graduated students is small, is the fact that there are also students who start their study, but quit and therefore do not finish their study. At this moment these students are marked as students who are still studying (since they did not graduate). This is because it would be incorrect to include these students in the graduated group, since this would pollute the success rates. In an ideal data set these students are removed from the set. It is however not easy to extract these students from the dataset, since the information that they quit was not included in the data set. The only way to do so is manually inspect the students and decide if they are still studying based on their last exam trial. Doing this would mean that students are excluded based on a decision which is quite disputable, so this is not done.
When the group of graduated students is sparse and there is a substantial amount of constraints (around ten or more), another problem might occur. In this case there are 2^{11} number of different constraint violation combinations, which is equal to 2,048 different combinations. The number of possible combinations outnumbers the number of available students by a lot. This has the effect that for a lot of combinations of violated constraints only a few students are marked as similar. The effect would especially occur for the suggestions, since unusual actions might be performed during trace execution. If a category contains only a few similar cases, the suggestions can be unreliable.
Another effect of the large amount of states is that the dashel containing all the combinations of constraint violations becomes unclear, there are a lot of combinations and for most of these combinations there are only a few similar students. So if the number of constraints increases this visualization becomes less useful.

The last problem in the data is specifically caused by the type of constraints and the available timestamps. The type of constraints used in this chapter, A before B means that course A should be completed before course B is started. In practice this means that course A should at least be completed in the quartile before the quartile in which course B starts. It is however possible to follow the courses A and B in parallel. According to the previous definition the constraint A before B is violated, but if the exam of course A is before the exam of course B and the exam of A is passed, the constraint is not violated with the current implementation. This is due to the fact that in the data set only exam dates are available and no other dates at all. This means it cannot be checked when the course is followed by the student. Therefore, in this case study it is accepted if the courses are followed in parallel, but the exam of course A is before course B.

7.4 Discussion of the results

In this section the results of this case study are discussed. In Section 7.4.1 the influence of the prior-knowledge constraints is discussed. Section 7.4.1 is dedicated to the suggestions for the next course to continue the master.

7.4.1 Discussion of the prior-knowledge constraints

The results are shown in Figures 7.2, 7.3 and 7.4 and also in Table 7.2. For the *In Time* criterion the difference in success rate is significant, about 55 % against 37 %. Both these groups (violated and non-violated) are also based on a significant number of students, 215 and 142. This shows us that it pays off to follow the study guide as designed by the head of

the program. Which is good to know.

Prior-Knowledge	Success Rate	Pass #	Fail #	Total #
Non Violated	54,9 %	78	64	142
Violated	36,7 %	79	136	215
Constraint 1	31 %	40	89	129
Constraint 2	36,8 %	35	60	95
Constraint 3	28,3 %	26	66	92
Constraint 4	40,7 %	33	48	81
Constraint 5	-	0	0	0
Constraint 6	20,6 %	14	54	68
Constraint 7	35,3 %	6	11	17
Constraint 8	52,0 %	13	12	25
Constraint 9	43,8 %	7	9	16
Constraint 10	30,8 %	4	9	13
Constraint 11	43,5 %	50	65	115

Table 7.2: Results for the *In Time* criterion

The constraints are also shown separately in the table. When the distinct students of these constraints are taken together, you have the same number of students as in the violated case. The strong constraints from the program are again in bold text. This should have meant that the success rate for the strong constraints should be much lower than for the weak constraints. This is however not the case for all these constraints. The average success rate for any constraint is about 37 %, but only constraints 1, 3 and 10 of the strong constraints are lower. From the weak constraints, 6 is dramatically lower than the other weak constraints while it is still based on a reasonable number of students. This would insinuate that constraint 6 should be a strong constraint instead of constraint 2. Number 7 is also lower, but this one is based on much lower number of students.
Another notable result is that students who violate constraint 8 (APM before SAIS) do not perform worse than the students who do not violate any constraint. This might mean that this constraint can be removed at all. The last notable result is that constraint 5 is not even violated once, so nothing useful can be said about this constraint.
For the second criterion, *Delay*, the results are about the same (as expected). Noteworthy is that the success rates of the constraints which performed bad for the *In Time* criterion improve more than rates of the other constraints. All the other statements still hold for this criterion. The results are shown in Table C.1.

7.4.2 Suggestions for next courses

A lot of the suggestions for this case study suffer from the fact that there are not enough minimal cases and are therefore marked as unreliable. If the minimum of similar students is set to ten (which is equal to 2,8 % of the total number of finished cases), it is impossible to give a reliable success rate for several suggestions per student and if it is increased to about 5 % even more suggestions are imprecise (not enough similar students).
Checking several students with different parts of the program completed, the most regular top

suggestions are BIS, BPMS and MI. For the first two courses this is as expected, since these courses are marked as prior-knowledge for a lot of other courses. The course MI however is a bit strange, because if you look at Figure C.1 it has the same prior-knowledge constraints as APM and BPS, so it is expected that the results are more or less the same.
If the students with no courses are inspected the difference is even bigger. Starting with MI results in a success rate of 49 % while for BPS it is only 18 %. This might mean that the constraints for BPS are more important than for MI.

7.5 Interpretation of the results

The context of this thesis comprises a method to gain insight in protocols used in processes. In this case study, the protocol at hand consists of prior-knowledge constraints. The interpretation of the results is beyond the context of this thesis. As a TU/e student who followed all the courses in the constraints, it is however possible to make some meaningful interpretations of the results. In this way the usefulness results can be verified.

According to the results discussed in Section 7.4 the most influential constraints are (in order) 6, 3, 10 and 1. The constraints BPMS before BPS, BIS before BPS, BPMS before SAIS and BIS before BPMS. As a student I can strongly relate to these constraints. BIS is essential to take as the first course in this program. Though the material taught in this course is not that complicated, you get familiar with all the concepts and terms in the process mining field. For both BPS and SAIS it is convenient that the course BPMS is completed first. In my program I first completed BPS before BPMS. Though I successfully completed BPS in one try, some of the material taught in BPMS would be convenient to know beforehand.
According to the visualization in Figure 7.2, only the constraint APM before SAIS has no influence (violating this constraint results in the same success rate as not violating it). In my own experience this is not true, since a lot of the material treated in the course APM is also used in the material in SAIS. This was not the case for MI for example. However, the influence is still strong for that course.
In my opinion, one important aspect is missing in this data. To complete the master program in two years it is necessary to complete the graduate project in six months. This is often not the case, and one or two (or even three or more) months extra are necessary to complete the project. Often it is possible to make up for some failed exam trials while the student does not lose time because of the retrials in the quartiles after the original trial. This is not possible for the graduate project, except if the student is super smart and completes the courses in the master in 4 or 5 quartiles instead of the original 6. In this case study it would be very useful if an extra constraint (complete graduate project in six months) was added such that this influence can be measured. Unfortunately, this data is not available.

As mentioned in Section 7.4.2, the interpretation of the suggestions is a lot harder, due to the fact of the small set of closed cases. However still some useful results can be derived. As mentioned in the introduction of this chapter, the head of the AIS department was interested in the influence of the prior-knowledge constraints, such that they can be taken into account when courses are assigned to a quartile. From these results it is wise to make sure that the course BPMS is given as early as possible while a course as BPS is given further on the program, such that prior-knowledge constraints are met. For the head of the department

it might be a relief, that according to the results BPMS does not have as much influence on MI as expected. This means that the scheduling of MI does not give as much conflicts as might be expected. For APM no real interpretations can be done, because there is simply not enough data available.

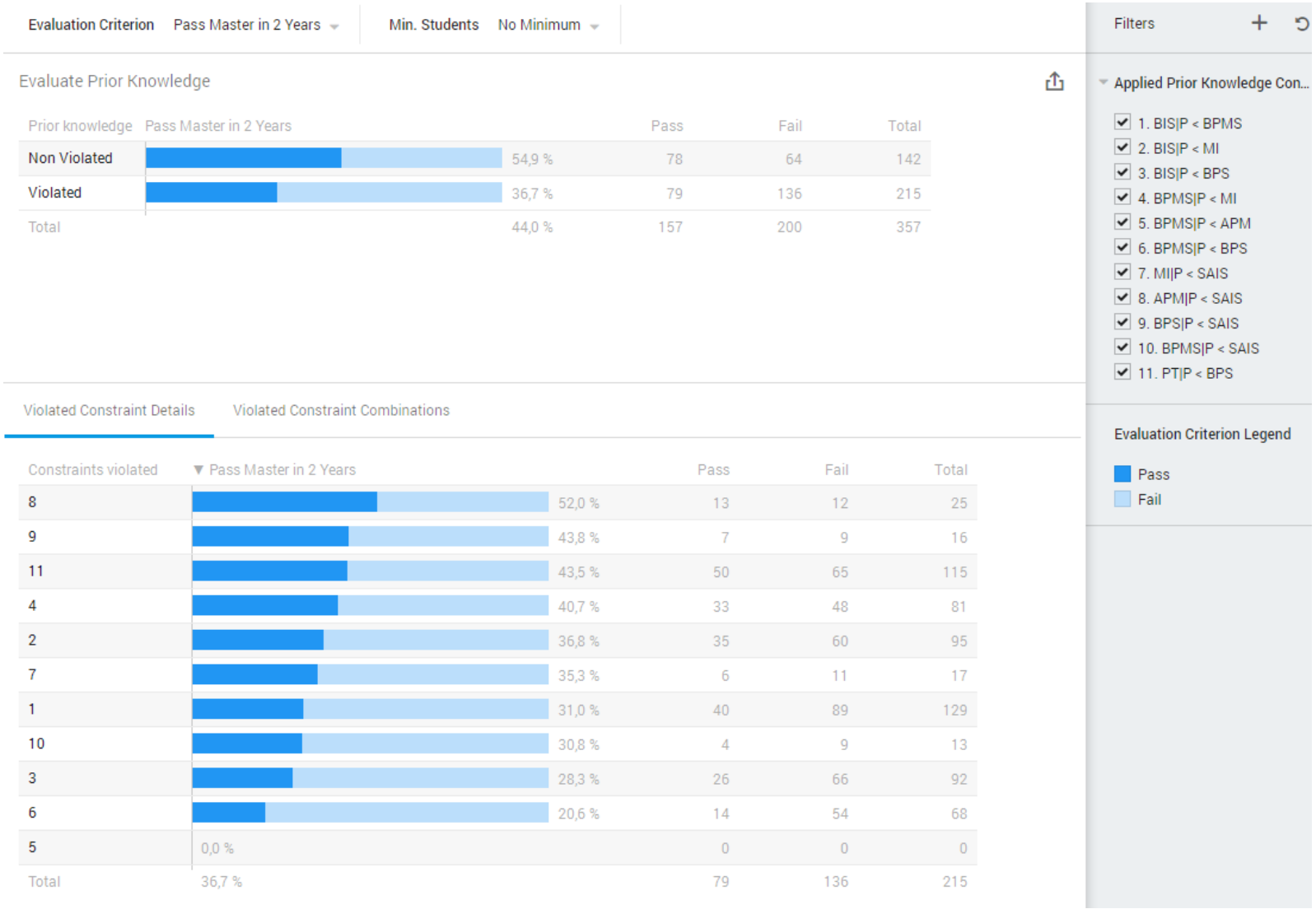


Figure 7.2: Overview of the effect of the prior-knowledge constraints for *In Time*, only single constraint details

Figure 7.3: Overview of the effect of the prior-knowledge constraints for *In Time*, all violation combinations

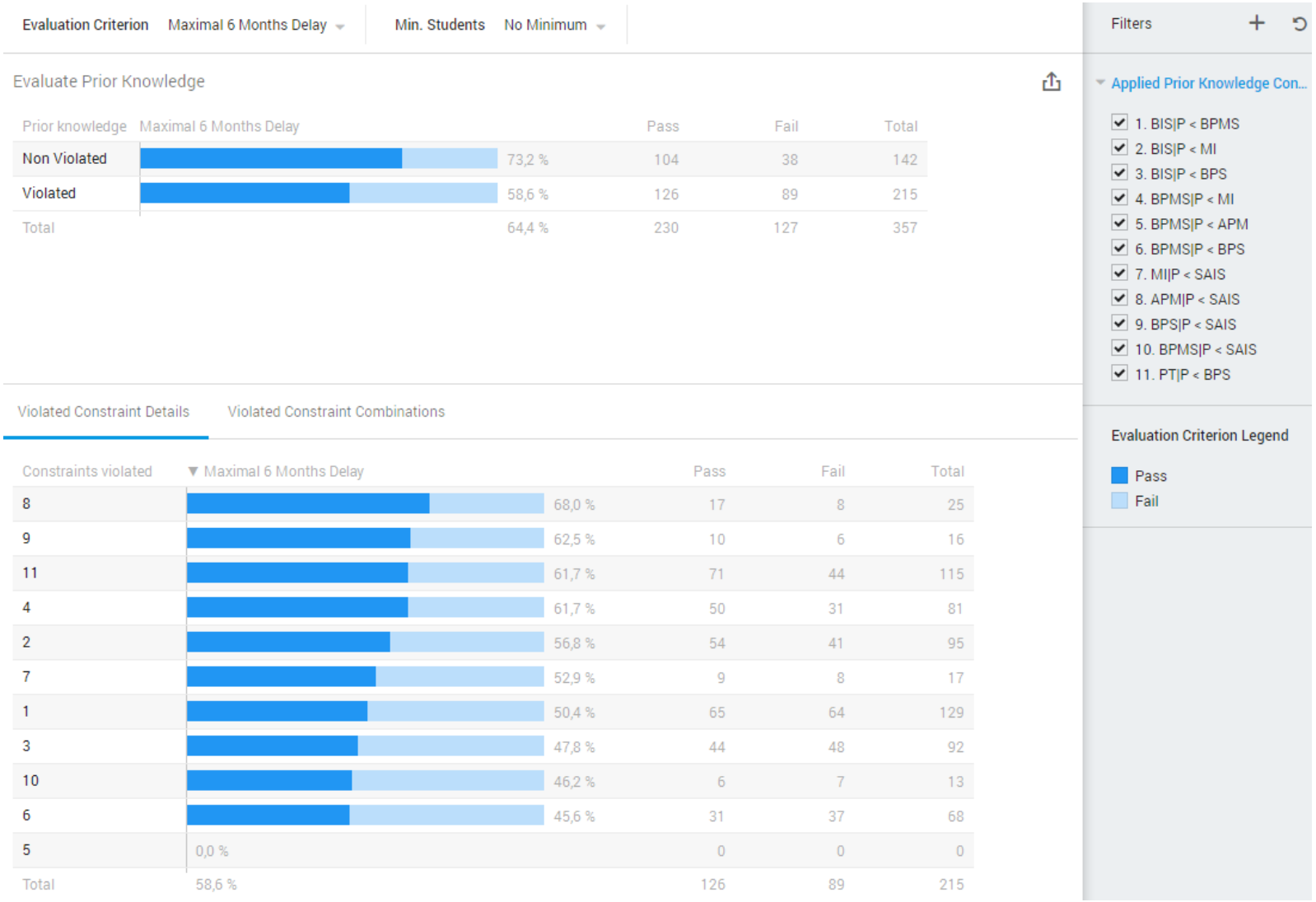


Figure 7.4: Overview of the effect of the prior-knowledge constraints for *Delay*, only single constraint details

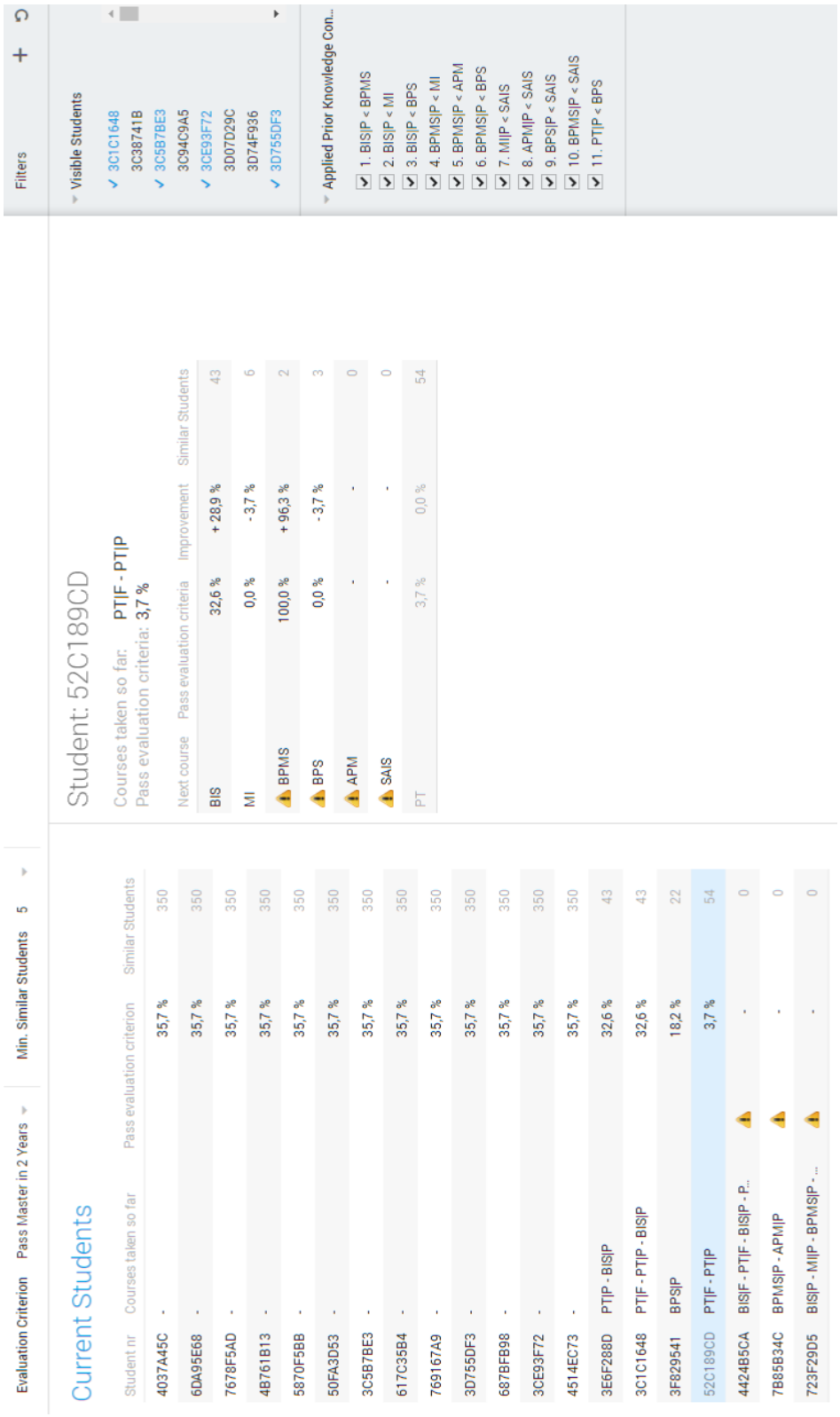


Figure 7.5: Overview for suggestions for new courses, for a selection of students

Chapter 8

Real-Life Hospital Process Case

In this chapter the second case study is analyzed. The process to be analyzed is the process of an anonymous hospital in the Netherlands. Compared to the previous case study, the difference is that processes in a hospital are less structured and less clear than the courses process. The process is also more influenced by external factors. In Section 8.1 the process is described. This description is formalized in Section 8.2. Section 8.3 indicates possible problems related to the quality of the dataset and the nature of the process. The results are discussed in Section 8.4. The chapter concludes with a discussion of the interpretation of the results in Section 8.5.

8.1 Description of the hospital process

Processes in hospitals are complicated and usually consist of multiple sub-processes. The hospital treated in this case study is no exception to this rule. Due to the fact that the process in the hospital as a whole is very complicated and too large for this case study, the process to be analyzed is a sub-process within the hospital. This sub-process is the process of treating patients with a *Community Acquired Pneumoniae*, further on referred to as CAP. In plain English this means treating patients who suffer from an infection in the lungs, which was acquired outside of the hospital.
The reason this sub-process is chosen, is that in the Netherlands, there are guidelines[1] on how these patients should be treated. These guidelines contain both strict rules, which should always be followed, and advice, which does not have to be taken very strict. Therefore, a subset of nine rules is abstracted from this protocol.

1. When Influenza is established (either Oral or Intravenous, for short IV) antibiotics for Influenza should be given.
2. When Streptococcus Pneumoniae is established (either Oral or IV) antibiotics for Streptococcus Pneumoniae should be given.
3. When Staphylococcus Aureus is established (either Oral or IV) antibiotics for Staphylococcus Aureus should be given.
4. A blood sample should be taken before any antibiotics are given.
5. A sputum sample should be taken before any antibiotics are given.

[1] SWAB Guidelines

6. When severe CAP is established, an antigen Urine sample should be taken to test for Streptococcus Pneumoniae.
7. When severe CAP is established, an antigen Urine sample should be taken to test for Legionella.
8. When severe CAP is established, the treatment should start with IV antibiotics.
9. At least two blood tests should be taken per patient.

To understand these rules, additional information may be necessary. A CAP can have three different levels. The first level is *Mild*. Patients with mild CAP can usually be treated from home and are therefore of less interest. The second level is *Moderate*. Patients with moderate CAP are treated in the hospital, but on a non-intensive care ward. The last level is *Severe*. Patients with severe CAP are very sick and are treated on the intensive care.

Next to these nine rules, an additional rule is added. This rule is not only applicable for lung infections and is based on bureaucratic reasons. When a test is done on a specimen of patients, the analyst is looking for isolates, which are micro-organisms that could indicate an illness. In the Netherlands it is only possible to charge the insurance three isolates per test. Therefore, the extra rules "Each test has at most 3 isolates" is added.

The head of the infection prevention department of the hospital wants to know if following the protocol has any influence on the possibility that a patient acquires a *Hospital Acquired Infection*, further on referred to as HAI. A HAI is an infection which is acquired during the stay in the hospital. The infection prevention department of the hospital is very focused in decreasing the number of patients with a HAI. When a patient acquires a HAI, it means that not the correct precautions were taken to prevent this infection. Patients already are more susceptible to infections, due to their illness. A HAI has big influence on the duration of the intake in the hospital and the costs of the intake. The head of the infection prevention department is interested if there is a correlation between this protocol and the possibility that a patient acquires a HAI.

8.2 Formalization of the problem

With the information available, the formalization needs to be applied. An event log containing 618 intakes in which the patient was diagnosed with either moderate or severe CAP is available. Of these intakes, 616 are already dismissed from the hospital and 2 are still ongoing. For all intakes a lot of information is available, but the traces have to be created ourselves. The traces are defined on intakes and contain only the actions relevant to the rules discussed in Section 8.1.
The fact that traces are defined on intakes did cause a problem for the formalization of one rule. The extra rule, maximal 3 isolates per test, has to be defined on test level to function correctly. In theory it is still possible to formalize this rule to a constraint in FLTL form, where it is defined on trace level. In practice this could not be done due to the limitations of Declare, where it is not possible to access other data attributes. Therefore, this rule is not formalized. It is still possible to measure the result of this rule for closed cases in MagnaView, though this way it is not based on the trace. Based on these reasons, this rule is only shown in the evaluation of the protocol and is not taken into account for the predictions.

The other rules can be formalized; however, the rules are altered a bit. Taken the rules very strict, will result in a lot of falsely accused fails for cases. The problem is in the establishment of micro-organisms. Doctors often treat their patients based on the symptoms shown. So if a patient shows the symptoms of influenza, he is given antibiotics against influenza, even if it is not yet officially established that the patient has influenza. The official establishment takes time, since first a sample has to be taken and from this sample isolates have to be grown. If the doctors have to wait on the official establishment every time a patient has a disease, this would result in a congested hospital and probably more dead patients, since they were not treated in time.
Therefore, the choice is made that if somewhere during an intake a relevant micro-organism is established, it is acceptable if the patient received medication against this micro-organism during this intake. This results in the following formulas, where the **U** is the weak until operator.

1. $\neg \Diamond$ Establish Influenza $\vee \Diamond$ Influenza Oral Treatment $\vee \Diamond$ Influenza IV Treatment
2. $\neg \Diamond$ Establish S. Pneumoniae $\vee \Diamond$ S. Pneumoniae Oral Treatment $\vee \Diamond$ S. Pneumoniae IV Treatment
3. $\neg \Diamond$ Establish S. Aureus $\vee \Diamond$ S. Aureus Oral Treatment $\vee \Diamond$ S. Aureus IV Treatment
4. $\neg$ Give Antibiotics^1 **U** Take Blood Sample
5. $\neg$ Give Antibiotics^1 **U** Take Sputum Sample
6. ($\neg \Diamond$ Establish Severe CAP $\vee \Diamond$ Take Antigen Urine Test for S. Pneumoniae)
7. ($\neg \Diamond$ Establish Severe CAP $\vee \Diamond$ Take Antigen Urine Test for Legionella)
8. $((\neg \Diamond$ Establish Severe CAP$) \vee ((\Diamond$ Give IV Antibiotics$^2) \wedge (\neg \Diamond$ Give Oral Antibiotics$^3)))$ $\vee (((\neg$ Give Oral Antibiotics$^3)$ **U** Give IV Antibiotics$^2) \wedge (\neg \Diamond$ Establish Severe CAP$)$ $\vee (\Diamond$ Give IV Antibiotics$^2)))$
9. $\Diamond$ Take Blood Sample $\wedge \Diamond$ Take Blood Sample

For ease of reading, activities are grouped together. These grouped activities are denoted with a number on the right corner.
1 = {Give Influenza Antibiotics IV, Give Influenza Antibiotics Oral, Give IV Antibiotics, Give Other IV Antibiotics, Give Other Oral Antibiotics, Give S. Aureus Antibiotics IV, Give S. Aureus Antibiotics Oral, Give S. Pneumoniae Antibiotics IV, Give S. Pneumoniae Antibiotics Oral}
2 = {Give Influenza Antibiotics IV, Give IV Antibiotics, Give Other IV Antibiotics, Give S. Aureus Antibiotics IV, Give S. Pneumoniae Antibiotics IV}
3 = {Give Influenza Antibiotics Oral, Give Other Oral Antibiotics, Give S. Aureus Antibiotics Oral, Give S. Pneumoniae Antibiotics Oral}

The relevant evaluation criteria also need to be retrieved from the problem description. In this case study the hospital is only interested in the effect on HAI. Therefore, only one criterion was defined.

1. **The patient does not have a HAI during his intake.**

This criterion shall be referred to as *No HAI* in the remainder of this chapter. The information to determine this per patient is already present in the event log. The automata associated to each of the constraints are shown in Appendix D

8.3 Data quality

As in the previous chapter, the sparsity of the available cases is a problem. From the past 5 years, 618 patients with a moderate or severe CAP had an intake in the hospital. This small set is caused by the fact that the hospital is not very large and the good health care system in the Netherlands. Lung diseases are often discovered in an early stage which affects the treatment. The number of people in the Netherlands with a moderate or severe CAP is not so high. The result is that due to the sparse set, there will be categories with no or only a small amount of similar cases.
Another potential problem is that the process in a hospital is influenced by a lot of external factors, not available in the data or covered in the protocol. There are various legit reasons to differ from the protocol. For example, the patient is in critical condition and the general tests are skipped, or the patient is resistant for some antibiotics. If these reasons are not covered in the protocol but are still legit, the compared categories might show distorted results.

8.4 Discussion of the results

In this section the results of the second case study are discussed. In Section 8.4.1 the influence of the constraints on the *No HAI* criterion is discussed. In Section 8.4.2 possible continuation actions are discussed for the open cases in the process.

8.4.1 Discussion of the constraints

The results of the influence of the constraints on the *No HAI* criterion are shown in Figure 8.1 and 8.2. The first insight which can be seen is that in almost all the cases one or more of the constraints are violated. Out of the 616 available cases, 612 of them violate the constraints and only 4 cases abide by all these constraints.
The second thing which stands out, is that the success rate of the violated cases seems very good (about 90 %) at first. When you put this result in perspective, this means 1 out of every 10 intakes acquires an infection which was caused by something in the hospital. This sound a lot worse than the 90 % score. This is also the reason that the head of the infection prevention department is interested in how to prevent more of these infections.
When the influence of the constraints on their own are investigated one particular constraint really stands out. This is constraint number 6, "*Maximal 3 isolates per test*". For the intakes violating this constraint, the success rate is only 54 %. There might be reasonable explanation for this. It is more likely that this violation is caused by the fact that for an intake a HAI is established, than the other way around. When a patient has a HAI, he is very sick and his stay in the hospital is longer than normal. Due to the sickness it is likely that more micro-organisms are found and more tests are necessary. When you combine this explanation with the two facts that only 2 % of all the cases violate this constraint and that this constraint is based on a financial basis, it is safe to say that there is probably no correlation between this constraint and the likelihood that a patient acquires a HAI.

Next to constraint number 6, three other constraints stand out because they differ a lot from the average success rate. These constraints are number 7, 8 and 9. All of them have a success rate between the 65 and 70 % and all of them involve establishing a severe CAP. All three constraint categories are based on enough cases to make them relevant. Since all

constraints which involve establishing a severe CAP differ from the average success rate, this insinuates that there is a connection between the fact that a patient acquires a severe CAP and the likelihood that a patient acquires a HAI.
It is easy to say that it would probably be better for the hospital to monitor the adherence to these constraints more, but this is not always possible in a hospital environment. In a hospital a lot of external factors weigh in, in the decision to take a test or give a certain medicine. Most of these external factors cannot be taken into account by the protocol. Especially for these three constraints. Patients with a severe CAP are very sick and it is not always possible to conform the protocol at all points, due to a lack of time or because the patient is just too sick to do a large number of tests. Therefore, it is not straightforward to say that these constraints should be regulated more.
The fact however that all three constraints concerning severe CAP have a worse success rate than the other constraints gives an insight. The protocol probably needs some refinement for these patients, since they are more likely to acquire a HAI. The violations combination visualization supports this statement, but does not give any new insights.

8.4.2 Suggestions for next actions

The open cases can be seen in Figure 8.3. There are only two intakes where a patient has a CAP. Unfortunately, one of the two intakes has 0 similar cases and all of the possible continuation actions also have 0 cases. Therefore, no meaningful predictions can be made for this intake. This means only the other intake shall be handled in this section. As can be seen in Figure 8.3 this intake is selected and has a current success rate of 91 % to not acquire a HAI.
On the details dashel in the visualization all the relevant actions are shown. What immediately stands out is that about half of these actions would result in not enough similar cases (note that the minimum is set to 5 cases). All the other actions have the same result, which is that the success rate drops with 2 %. This seems strange, but it is caused by the fact that some of the successful cases took the actions *Take Blood Sample* or one of the actions involving *S. Aureus Antibiotics*. This results in smaller groups, where only the group remaining in the same state has enough cases to be seen as reliable. For this intake it is not clear which action is the best continuation action. None of the actions stand out, since the actions with enough cases do not improve the success rate and the actions which improve the success rate have not enough similar cases.
It is noteworthy to say that compared to the previous case study, the result of this prediction should be seen even more as an advice than in the previous study. This is caused by the fact that this protocol does not incorporate all the external factors in a hospital. For this sort of complex processes with a lot of external factors it is very difficult to predict next continuation actions.

8.5 Interpretation of the results

Compared to the first case study, it is not possible to give an interpretation of the results for this process. The reason is that in contradiction to the first study, hardly any knowledge about the process is available. It is difficult to interpret the results if the context in which results need to be interpreted is unknown. The lack of knowledge about the process causes more possible problems for the results.

First, it is not possible to create a useful evaluation criterion if it is not known what the process tries to achieve. The experts from the hospital are interested to gain insights in hospital acquired infections. Therefore the criterion *No HAI* is created, but it is not possible to deduce any other relevant criterion. Even more difficult is to extract the relevant constraints from the protocol. Many information recorded in the protocol cannot be measured, since it is not specifically defined what has to be done or the data to measure is not stored. An example constraint which cannot be evaluated is *"P. aeruginosa should be considered in patients with severe structural lung disease and CAP"*. For this constraint it is not defined how to measure if something is considered or not. Most of the constraints suffer from issues like this. It does not help that the protocol consists of guidelines for hospitals in the Netherlands in general, instead of a detailed set of rules for one specific hospital.
A point of discussion is the possible correlation between the constraints and the criteria. In the first case study this correlation is present. To pass the master in two years, it is important to pass the courses and the constraints concerned the requirements to pass the courses. For this study the correlation between the constraints and the *No HAI* is less obvious. For example, it is very likely that whether a patient acquires a HAI is not influenced in any way by doing or not doing a Legionella test, when severe CAP is established. It is more likely that there is correlation between this criterion and some external factor, for example the fact that doctors do not always wash their hands after bathroom visits. The problem is that this last correlation cannot be tested, since the information is just not available.

It is clear that these sort of unstructured processes with a variety of external factors are maybe not very suitable to apply the method presented in this thesis. There are too many possible violations which are allowed in some cases, but these are not covered in the protocol. It may be considered to not use an established protocol, but to ask the specialists in the hospital if they can create the constraints relevant to the process in their hospital. This was not possible for this case study and the downside is that it requires a significant amount of time and extensive knowledge of the process at hand.

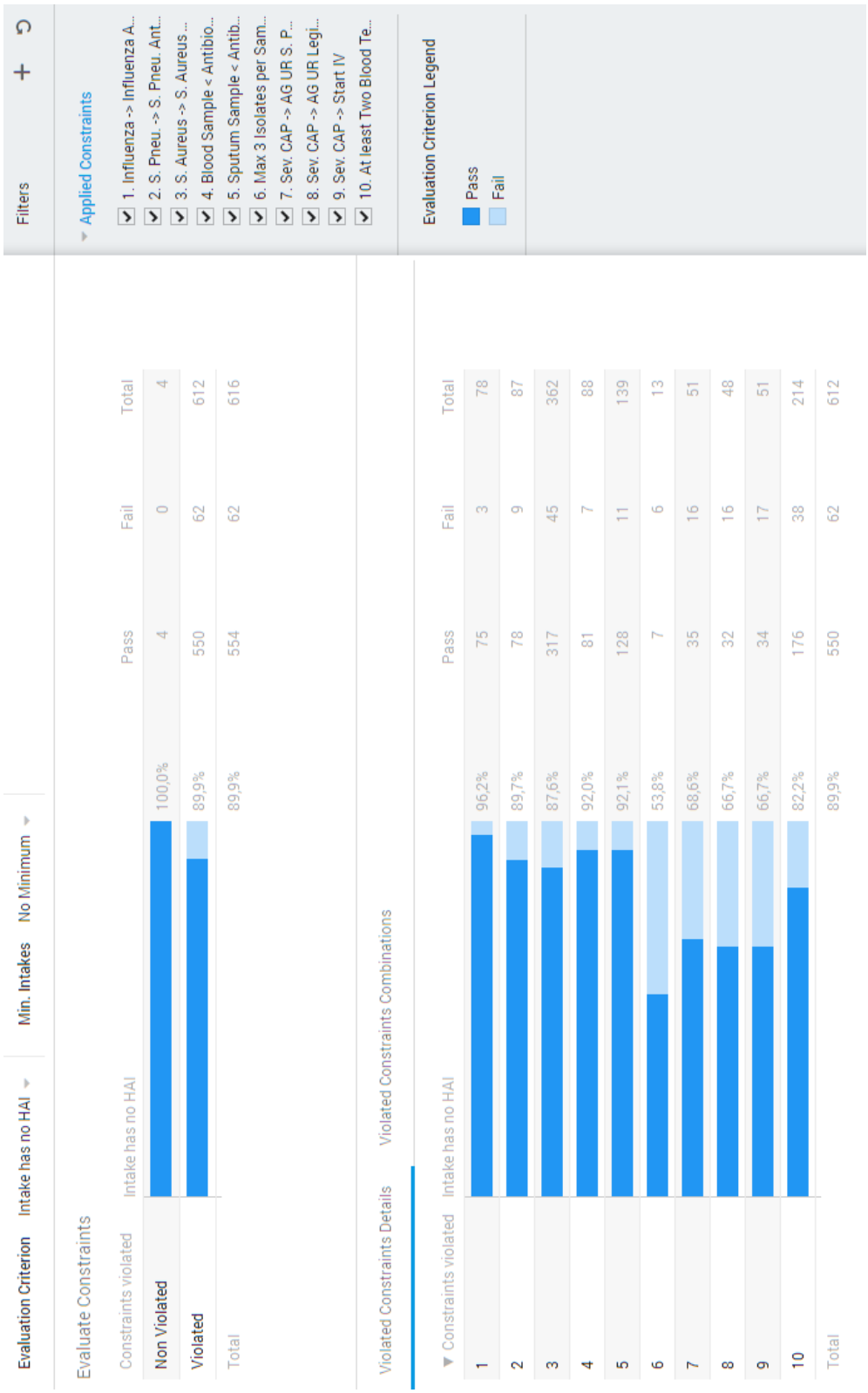


Figure 8.1: Overview of the effect of the constraints on *No HAI*, only single constraint details

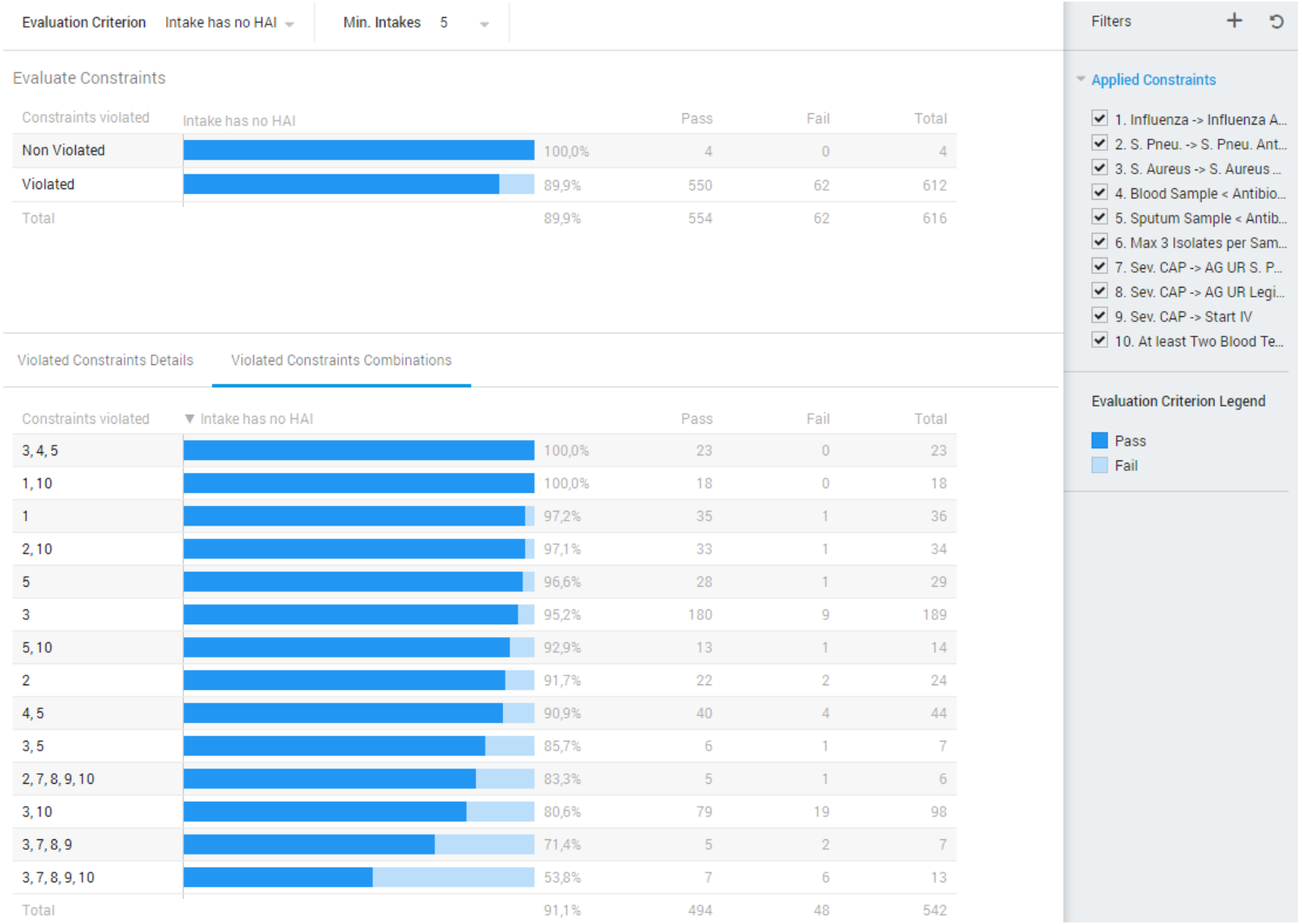


Figure 8.2: Overview of the effect of the constraints on *In Time*, all violation combinations with a minimum of 5 cases

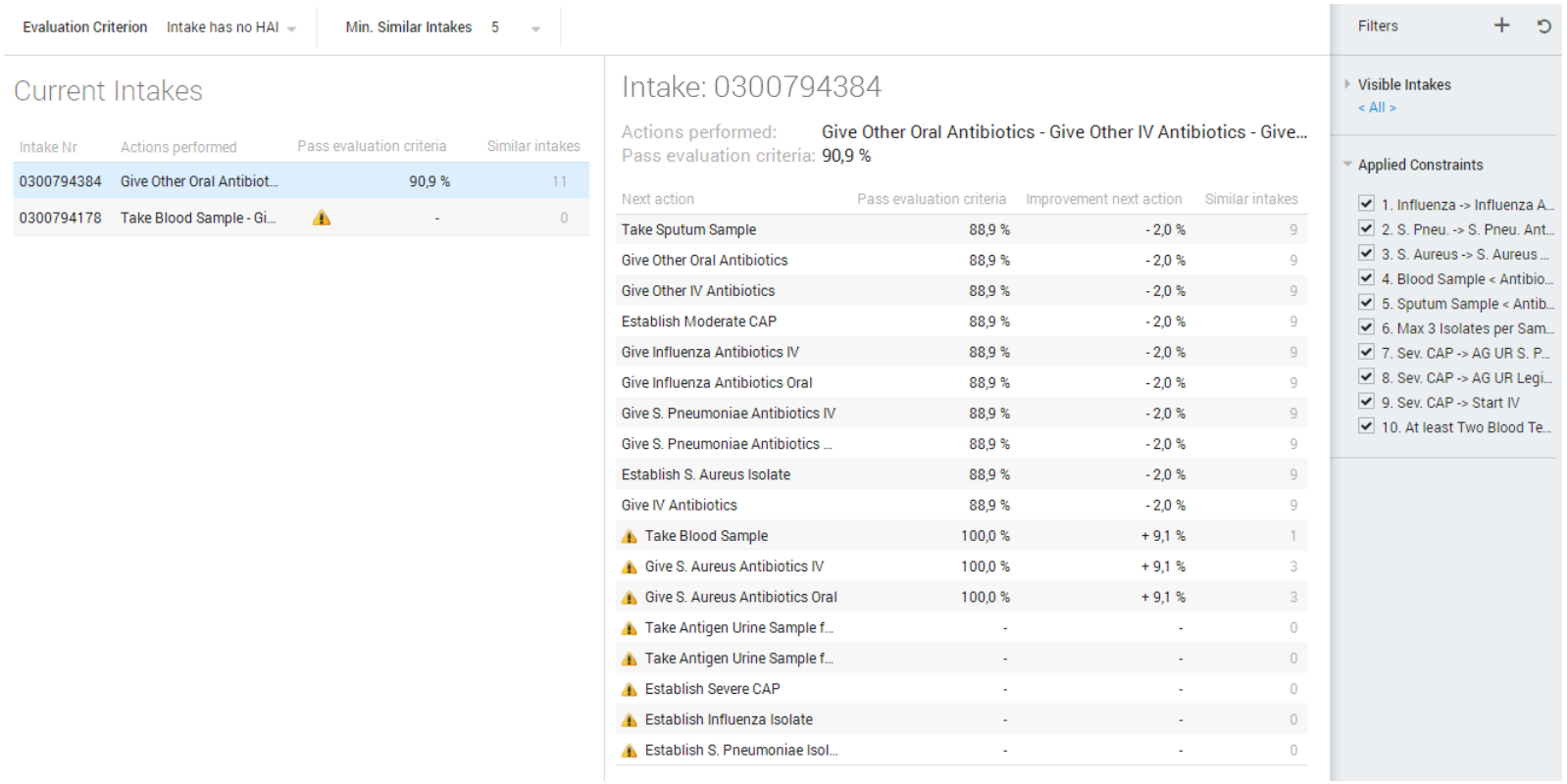


Figure 8.3: Overview for suggestions for new continuation actions for intakes

Chapter 9

Conclusion

In the business world, the way a process should be executed is often strictly defined. To keep up with the competition it is important to continually improve the process to cut costs. To know how the process can be improved it is important to know how the process performs. This can be achieved by evaluating processes by applying process monitoring. In this master thesis, an approach is developed to evaluate processes based on constraints. The result is the following research objective "*To develop an approach which evaluates a protocol, as a whole, or the parts within the protocol individually, on a given criterion and predicts the outcome for this criterion for open cases*". This objective is twofold, at one hand the protocol on which the process is based is evaluated to discover weaknesses in the protocol. At the other hand this evaluation is used to adjust open cases such that they are more likely to have a successful evaluation.

To achieve this objective, several research questions are deduced from the problem description. The first two questions are "*How can a protocol and an evaluation criterion be represented, such that it can be evaluated?*" and "*How can the influence of a protocol on an evaluation criterion be measured?*". The solution to these question is to use finite linear temporal logic to model the constraints deduced from the protocol. This has the advantage that for each activity it is strictly defined for a case whether the constraint is satisfied or not.
For the influence of constraints on the evaluation criterion, a metric is designed. This is the *success rate* metric. The success rate is defined on combinations of constraint violations, ranging from zero to all constraints violated. For each of these categories the success rate is defined as the chance a case has to have a successful evaluation in that category. This chance is based on the evaluation outcomes of the other cases in the same category. With this metric and the formal constraints, it is possible to evaluate the influence of the constraints on the evaluation criterion. Therefore, the first part of the research objective is achieved.

For the second part of the research objective the question "*How can the predicted outcome of an evaluation criterion for open cases be used to suggest the best continuation action during execution?*" is created. To answer this question, it is necessary to know what the predicted outcome is for an open case. To solve this, a definition is formed when a closed case is similar to an open case. For all similar cases, the success rate can be calculated and this is the predicted outcome for an open case. The suggestions are created with a recommendation system that uses these predicted success rate. For each open case the recommendation sys-

tem appends the case with a relevant action and calculates the success rate of this new state. The action which results in the highest success rate and has enough similar cases according to the user is given as suggestion to continue. With these suggestions the second part of the research objective is achieved.
It is important to present the result from this objective in a way such that the user can comprehend it. Therefore the last question "*How can the outcome of an approach be visualized, such that the result is intuitive for the user?*" is introduced. For both parts of the objective a visualization is created. Both these visualizations are evaluated. The users were able to interpret the results correctly and their feedback was incorporated in the visualizations.

Using the answers from these questions it can be concluded that an approach to evaluate a protocol and predicts outcomes for open cases is successfully developed. The evaluation of the two cases studies showed that the approach can be successfully applied on structured processes with a small number of external factors. The evaluation successfully shows which parts of the protocol have a large influence on the performance of the process. The suggestions presented are in line with the evaluation and recommended actions which should improve the success rate for the evaluation criteria.

When the approach is applied on loosely structured processes, where the influence of external factors is substantial, the approach is less successful to give insightful results. The evaluation suffers from many allowed violations of constraints, i.e. the constraint is violated but in a particular case this is allowed because the constraint does not cover all the edge cases. The suggestions are solely based on the protocol provided. In processes with external factors, the decision on how to continue the case execution is influenced by these factors. The suggestions cannot take these factors into account because the factors are not covered by the protocol.

The results of this approach depend heavily on the interaction between specialists with extensive process knowledge and the developer who does the formalization. The formalization requires knowledge of both developing with the MagnaView platform and writing formal FLTL formulas covering the constraints. It cannot be expected from the process specialists that this knowledge is known; it is most likely that this has to be done by a developer at MagnaView. The hospital case study however shows that the knowledge of the specialist is evident for a meaningful evaluation. The contradiction between the two studies concerning the meaningfulness of the insights is significant, due to the difference in knowledge about the process. Therefore, the quality of the insights will always be dependent on how good the collaboration between the developer and the specialists is.
Another point which needs to be taken in to account is that the approach is dependent on the amount of available data. Even to define a simple process, the number of constraints can easily grow. If the amount of available closed cases is small, this will result in categories which are not statistical relevant. Especially the suggestion part will suffer if the amount of closed cases is small.

9.1 Future work

Though the objective "*To develop an approach which evaluates a protocol, as a whole, or the parts within the protocol individually, on a given criterion and predicts the outcome for this criterion for open cases*" is achieved, there is still room for improvement and various opportunities for further research are available.
As discussed in Chapter 6, the Declare framework is used to translate the constraints in automata. For the context of this thesis the expressive power of Declare was satisfactory, but there is room for improvement. In Declare it is not possible to use data attributes, as in FLTL formulas. Since the processes in this thesis were quite small, it was still possible to define the events without the need for data attributes. For larger processes this is however not possible anymore.
It would be an improvement to incorporate the generation of automata from FLTL formula in the MagnaView expression language. This would eliminate the lack of access to data attributes. It would also automate the translation of automata in Declare to automata in MagnaView, which is a lot of work.

Possible opportunities for further research are already discovered in this thesis. First of all, the approach currently evaluated the existing constraints from the protocol. It would however be nice if also functionality is provided to discover new correlations between activities in a process. It is possible that an undocumented constraint is relevant to the process, but with this approach there is no way to verify this.
The approach is currently implemented as a near real-time solution. There is an important reason to not do this real-time. Every time the data is updated, a lot of calculations have to be redone and these calculations are costly in time. The data can be refreshed every night and the calculations can be made before the users start working such that the process to be monitored is updated every day. This is not possible anymore when it has to function real-time and further research to improve the calculation time is necessary.
A third improvement can be, to apply a more elaborate analysis to identify which constraints are relevant to specific cases. For example, take the courses process from the TU/e. If the protocol contains the constraint *Pass course A before course B*, but both the courses are not in the curriculum of the student it would be appropriate to not use this student to analyze the influence of the constraint. This filtering of cases is not trivial and would require extensive analysis of the data, but it would give better results for constraints.
There is room for improvement in the suggestions part. It would be helpful if not all the actions which are relevant to the process in general are considered for each case, but only the actions relevant to a specific case. This requires applying data mining or machine learning to discover when an action is relevant during a process execution.
It might be of use to create a more elaborate set of visualizations for the evaluation of a protocol. For this thesis the current set of visualizations were satisfactory to the problem, but it might be possible to retrieve more information from the protocol and the data. During the case studies, a visualization which could evaluate a single constraint at a time could be convenient. That is, the violated category contains all the cases violating the particular constraint and the non-violated category contains the rest of the cases. This is possible by disabling all the constraints but one in the constraints filter, but it would be faster and perhaps easier for the user to have a standard visualization for this comparison. Next to this new visualization, it would be interesting, if the evaluation criteria can be exchanged with

the constraints. By doing this, it is possible to investigate if the constraints influence the evaluation criteria, or if the influence is the other way around.
At last, it would be helpful if a larger case study to validate the results for the suggestions could be conducted. The suggestions could not really be verified in this thesis, since it was not possible in practice to apply these suggestions. This case study should cover a process with a relatively small number of external factors, such that the suggestions can also be applied. It is important that the process offers a large amount of historical data, such that the problem that a state has not enough similar cases appears less often. The process should also offer room to do these kind of evaluation, since not every process is suitable to test these studies, especially processes where the cases are people are not suited for these studies.

Bibliography

[1] Adriansyah, A. (2009), *Performance analysis of business processes from event logs and given process models*, Master Thesis, Eindhoven University of Technology. 6

[2] Günther, C. and Van der Aalst, W. (2007), *Fuzzy Mining - Adaptive process simplification based on multi-perspective metrics*, Business Process Management, Springer Berlin Heidelberg, pp 328 - 343. 5

[3] Leemans, S. Fahland, D. and Van der Aalst, W. (2013), *Discovering block-structured process models from event logs-a constructive approach*, Application and Theory of Petri Nets and Concurrency, Springer Berlin Heidelberg, pp 311 - 329. 5

[4] Weijters, A., Van der Aalst, W. and Alves de Medeiros, A. (2006), *Process mining with the heuristics miner-algorithm*, Tech. Rep. WP 166, Eindhoven University of Technology, pp 1 - 34. 5

[5] Weijters, A. and Ribeiro, J. (2011), *Flexible heuristics miner (FHM)*, Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on IEEE, pp 310 - 317. 5

[6] Rozinat, A. and Van der Aalst, W. (2008), *Conformance checking of processes based on monitoring real behavior*, Information Systems 33.1, pp 64 - 95. 5, 6

[7] Mannhardt, F., De Leoni, M., Reijers, H. and Van der Aalst, W. (2015), *Balanced multi-perspective checking of process conformance*, Computing, pp 1 -31. 5

[8] Mannhardt, F., De Leoni, M., Reijers, H. and Van der Aalst, W. (2015), *Measuring the precision of multi-perspective process models.* 5

[9] Rozinat, A. (2010), *Process mining: conformance and extension*, PhD Thesis, Eindhoven University of Technology. 5

[10] Van der Aalst, W. (2011), *Process mining: discovery, conformance and enhancement of business processes*, Springer Science & Business Media. 5, 6

[11] Few, S. (2009), *Now you see it: simple visualization techniques for quantitative analysis*, Analytics Press. 22

[12] Adriansyah, A., Van Dongen, B. and Van der Aalst, W. (2013) *Memory-efficient alignment of observed and modeled behavior*, BPM Center Report: 03-03. 7

[13] Adriansyah, A., Van Dongen, B. and Van der Aalst, W. (2011) *Conformance checking using cost-based fitness analysis*, Enterprise Distributed Object Computing Conference, Los Alamitos, CA, USA, pp 55 - 64. 6

[14] Van Dongen, B. (2007) *Process Mining and Verification*, PhD Thesis, Eindhoven University of Technology. 4

[15] Adriansyah, A., Sidorova, N. and Van Dongen, B. (2011) *Cost-based fitness in conformance checking*, International Conference on Application of Concurrency to System Design, Los Alamitos, CA, USA, pp 57 - 66. 6

[16] Cook, J. and Wolf A. (1999) *Software process validation: quantitatively measuring the correspondence of a process to a model*, ACM Transactions on Software Engineering and Methodology (TOSEM) 8.2, pp 147 - 176. 6

[17] Maggi, F., Westergaard, M., Montali, M. and Van der Aalst, W. (2011) *Runtime verification of LTL-based declarative process models*, Runtime Verification, Springer Berlin Heidelberg, pp 131 - 146. 7

[18] Van der Aalst, W., De Beer, H. and Van Dongen, B. (2005) *Process mining and verification of properties: An approach based on temporal logic* Springer Berlin Heidelberg, pp 130 - 147. 8

[19] De Beer, H. and Van den Brand, H. (2004) *The LTL checker plugins a (reference) manual* Eindhoven University of Technology. 8

[20] Pnuelli, A. (1977) *The temporal logic of programs*, Foundations of Computer Science, 18th Annual Symposium on IEEE, pp 46 - 57. 7

[21] Maggi, F., Montali, M., Westergaard, M. and Van der Aalst, W. (2011) *Monitoring business constraints with linear temporal logic: An approach based on colored automata*, Business Process Management, Springer Berlin Heidelberg, pp 132 -147. 7

[22] Van der Aalst, W., Pesic, M. and Schonenberg, H. (2009) *Declarative workflows: Balancing between flexibility and support*, Computer Science-Research and Development 23.2, pp 99 - 113. 11

[23] Bauer, A., Leucker, M. and Schallhart, C. (2010) *Comparing LTL semantics for runtime verification*, Journal of Logic and Computation 20.3, pp 651 - 674. 9

[24] Ciccio, C, and Mecella, M. (2015) *On the discovery of declarative control flows for artful processes*, ACM Transactions on Management Information Systems 5.4: 24. 7

[25] Westergaard, M. (2011) *Better algorithms for analyzing and enacting declarative workflow languages using LTL*, Business Process Management, Springer Berlin Heidelberg, pp 83 - 98. 9, 11

[26] Dimitra, G. and Havelund, K. (2001) *Automata-based verification of temporal properties on running programs*, Automated Software Engineering 2001, Proceedings. 16th Annual International Conference on. IEEE. 7, 11

[27] Zohar, M. and Pnuelli, A. (1995) *Temporal verification of reactive systems: safety*, Springer Science & Business Media, Volume 2. 8

[28] Eisner, C., Fisman, D., Havlicek, J., Lustig, Y., McIsaac, A. and Van Campenhout, D. (2003) *Reasoning with temporal logic on truncated paths*, Computer Aided Verification, Springer Berlin Heidelberg, pp 27 - 39. 9

[29] Bauer, A., Leucker, M. and Schallhart, C. (2006) *Monitoring of real-time properties*, Foundations of Software Technology and Theoretical Computer Science, Springer Berlin Heidelberg, pp 260 - 272. 9

[30] Pesic, M., Schonenberg, H. and Van der Aalst, W. (2007) *Declare: Full support for loosely-structured processes*, Enterprise Distributed Object Computing Conference, 2007, 11th IEEE International. 8

[31] Schonenberg, H., Weber, B., Van Dongen, B. and Van der Aalst, W. (2008) *Supporting flexible processes through recommendations based on history*, International Conference on Business Process Management, Springer Berlin Heidelberg, pp 51 - 66. 12

[32] Vanderfeesten, I., Reijers, H. and Van der Aalst, W. (2008) *Product based workflow support: dynamic workflow execution*, International Conference on Advanced Information Systems Engineering, Springer Berlin Heidelberg, pp 571 - 574. 12

[33] Baeten, J.C.M. (2010) *Models of Computation: Automata and Processes*, Eindhoven University of Technology, Department of Mathematics and Computer Science. 11

[34] Van der Aalst, W., Adriansyah, A., Alves de Medeiros, A., Arcieri, F., Baier, T., Blickle, T., Bose, J., Van den Brand, P., Brandtjen, R. and Buijs, J. (2011) *In Proceedings of the BPM Workshops. Lecture Notes in Business Information Processing Series, 99*, Process mining manifesto. 4

[35] Silva, A., Bonchi, F., Bonsangue, M. and Rutten, J. (2010) *Generalizing the powerset construction, coalgebraically*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Vol 8. 18

[36] Gansner, E. and North S. (2000) *An open graph visualization system and its applications to software engineering*, Software Practice and Experience 30.11, pp 1203 - 1233. 29

Appendices

Appendix A

Running Example Visualizations

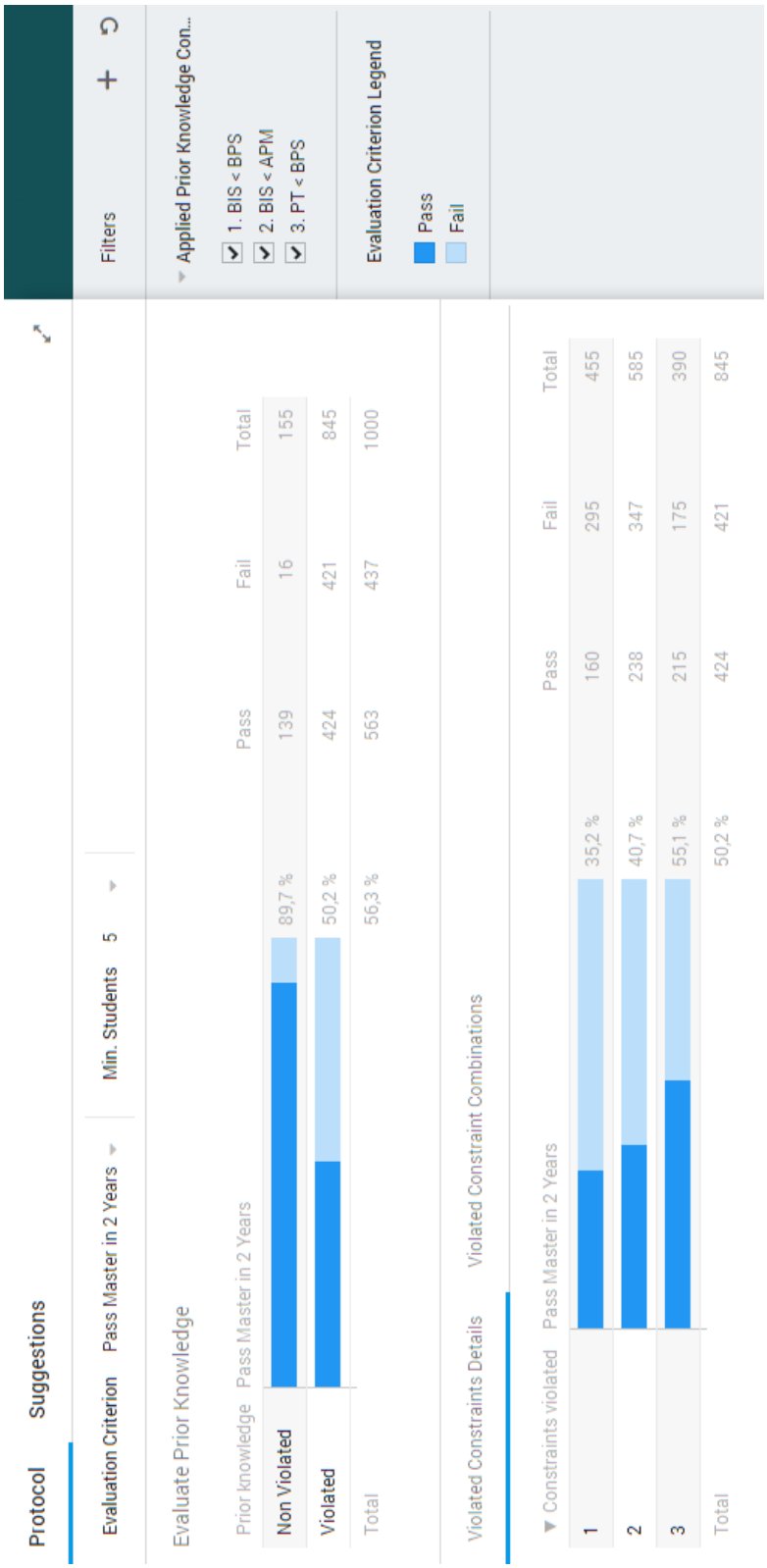


Figure A.1: Overview of the effect of the prior-knowledge constraints on *Master in 2 Years*

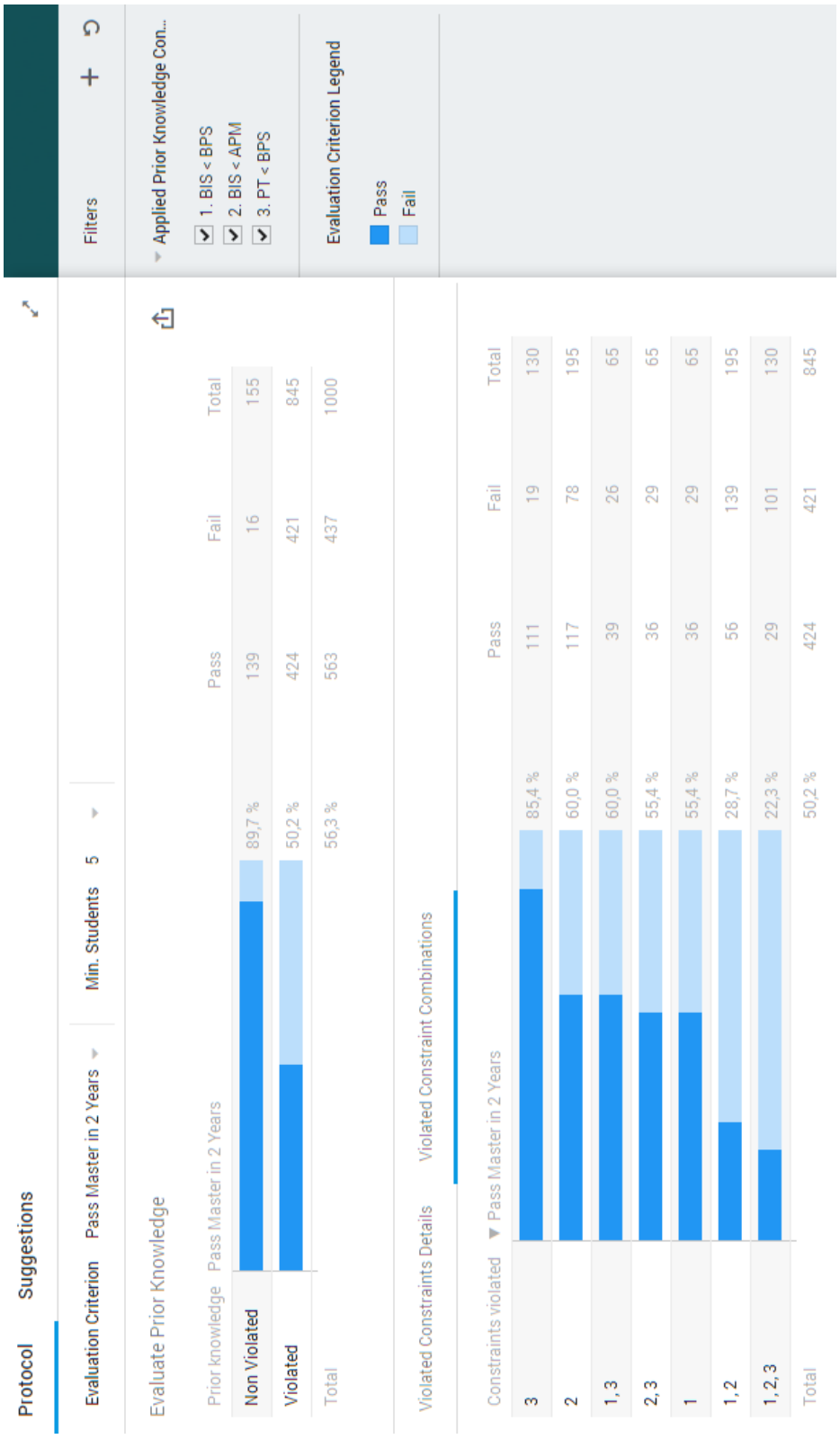


Figure A.2: Breakdown of possible combinations of violated constraints on *Master in 2 Years*

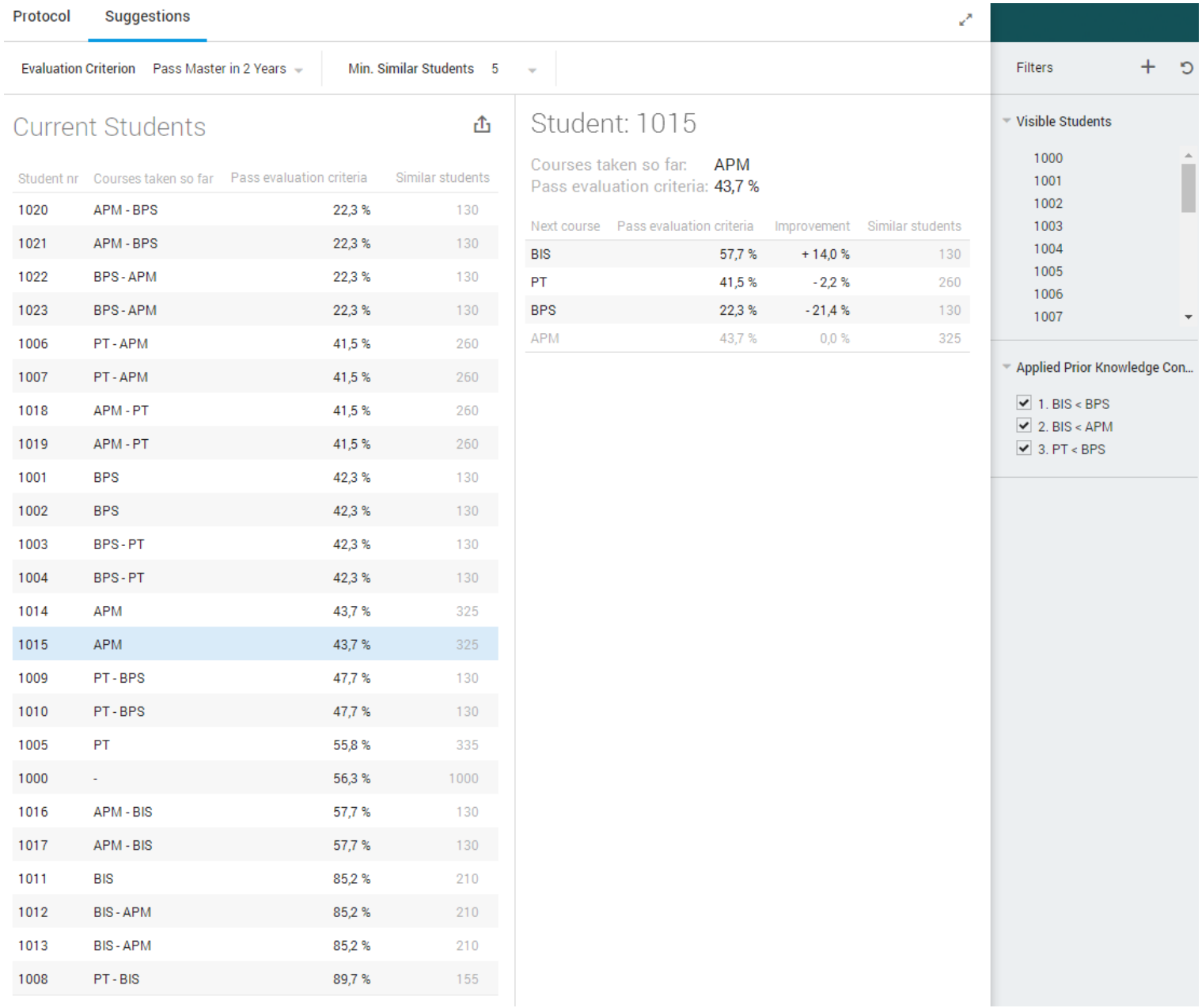


Figure A.3: Suggetions of a selected subset of students for the the *Master in 2 Years* criterion

Appendix B

Visualization Description

In this appendix, all the elements of the visualizations discussed in Chapter 5 are discussed more thoroughly. All the elements of the visualizations can be seen in Figure B.1, Figure B.2 and Figure B.3. The first 16 elements can be seen in Figure B.1.

1. The *Evaluation Criterion* filter. It is possible to choose one of the defined evaluation criteria with this filter. At this moment the criterion *Pass Master in 2 Years* is selected.
2. The *Minimum Cases* filter. In this filter the minimum number of cases can be selected. This minimum refers to the number of cases a category contains. For example, a particular constraint is violated by 15 cases. If the minimum is set to 20, this category will disappear. With this filter, categories containing not enough cases, and therefore might have unreliable results, disappear.
3. This dashel contains the overview of the *Violated* and *Non Violated* categories. In this overview the impact of these two categories on the evaluation criterion can be compared.
4. The *Violated* and *Non Violated* categories which are compared.
5. This bar chart shows the success rate of each category. The bars clearly distinguish which category performs better. This is due to the colors on the bars. The darker part denotes the successful cases and the lighter part the unsuccessful cases. To give detailed information, the rates are also expressed in a percentage next to the bars. Below the two categories, the success rate of all the cases together is also shown for comparison.
6. The number of cases per category which have a successful evaluation for the evaluation criterion.
7. The number of cases per category which do not have a successful evaluation for the evaluation criterion.
8. The total number of cases per category.
9. The constraints filter. In this filter the relevant constraints can be chosen. For example, if the user thinks that constraint 3 is not relevant, it can be unchecked. Only checked constraints are evaluated.
10. The legend defining the colors used in the bar charts.
11. The tab for the *Violated Constraint Details* dashel. By clicking on this tab, the associated dashel is shown. Out of the two possible dashels at the bottom, the associated dashel of this tab is shown by default.

12. This is the tab for the *Violated Constraint Combinations* dashel. By clicking on this tab, the associated dashel is shown.
13. The *Violated Constraint Details* dashel. In this dashel the influence of each constraint on its own is shown.
14. The constraint categories. Only the constraints which have enough cases are shown.
15. The success rate per constraint.
16. These are the same statistics as in the upper dashel, for all the constraint categories.

The following elements are shown in Figure B.2.

17. The same *Minimum Cases* filter. In this figure the minimum amount of cases is set to 5. This means that in the dashels at the bottom, all categories which have less than 5 cases are not shown.
18. The tab for the *Violated Constraint Combinations* dashel is currently selected. This is can be seen because this tab has a blue line below it.
19. The *Violated Constraint Combinations* dashel. With this dashel the combinations of violated constraints and their influence can be investigated.
20. These are all the violated constraint combinations categories. Note that in contradiction to the other dashel, a case can only appear in one category. Take for example the category 2, 4, 11. In the other dashel, the case appears in the three categories of constraints 2, 4 and 11, but in this dashel it only appears in this category.

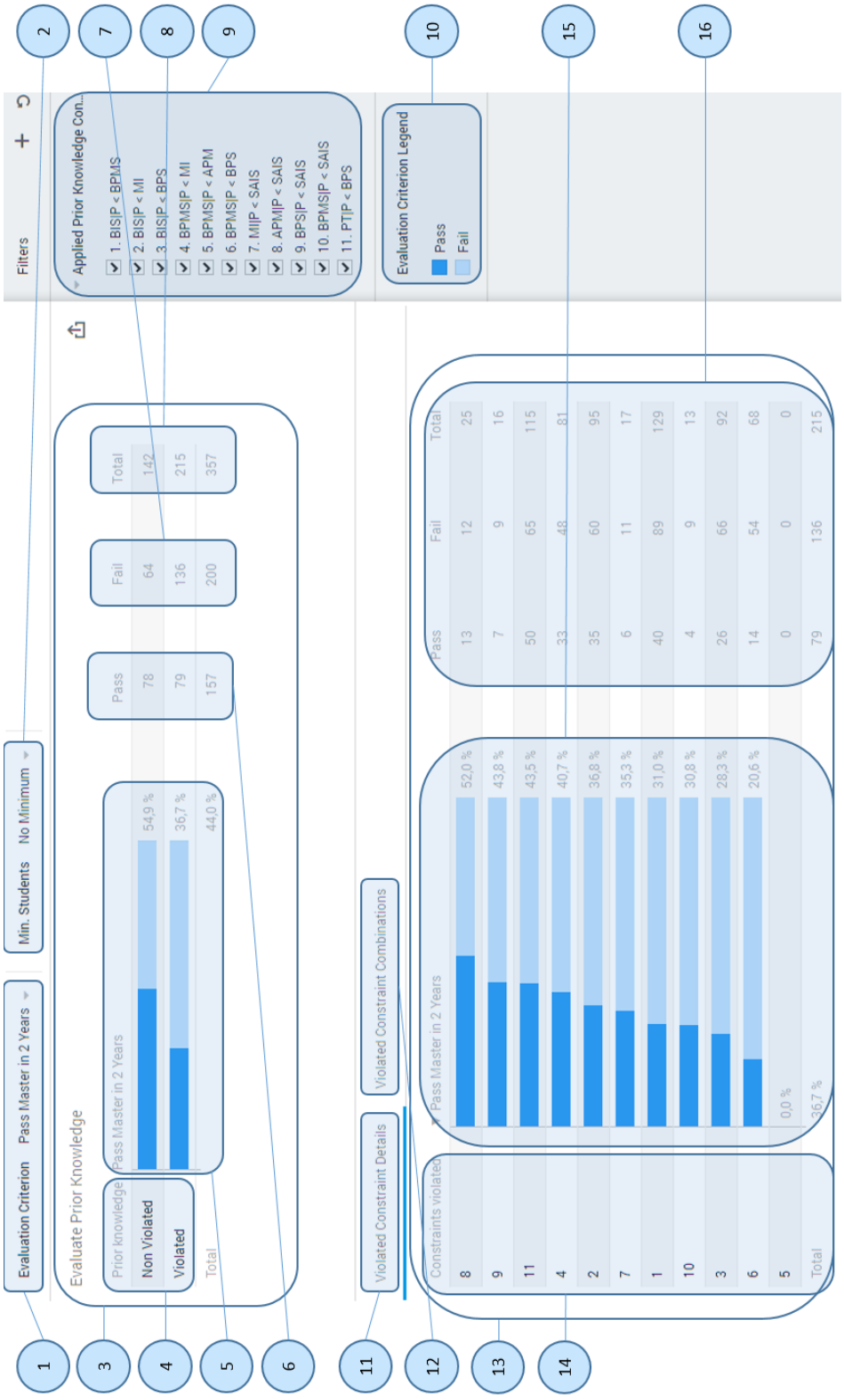


Figure B.1: Overview of the effect of the constraints on the evaluation criterion, only single constraint details

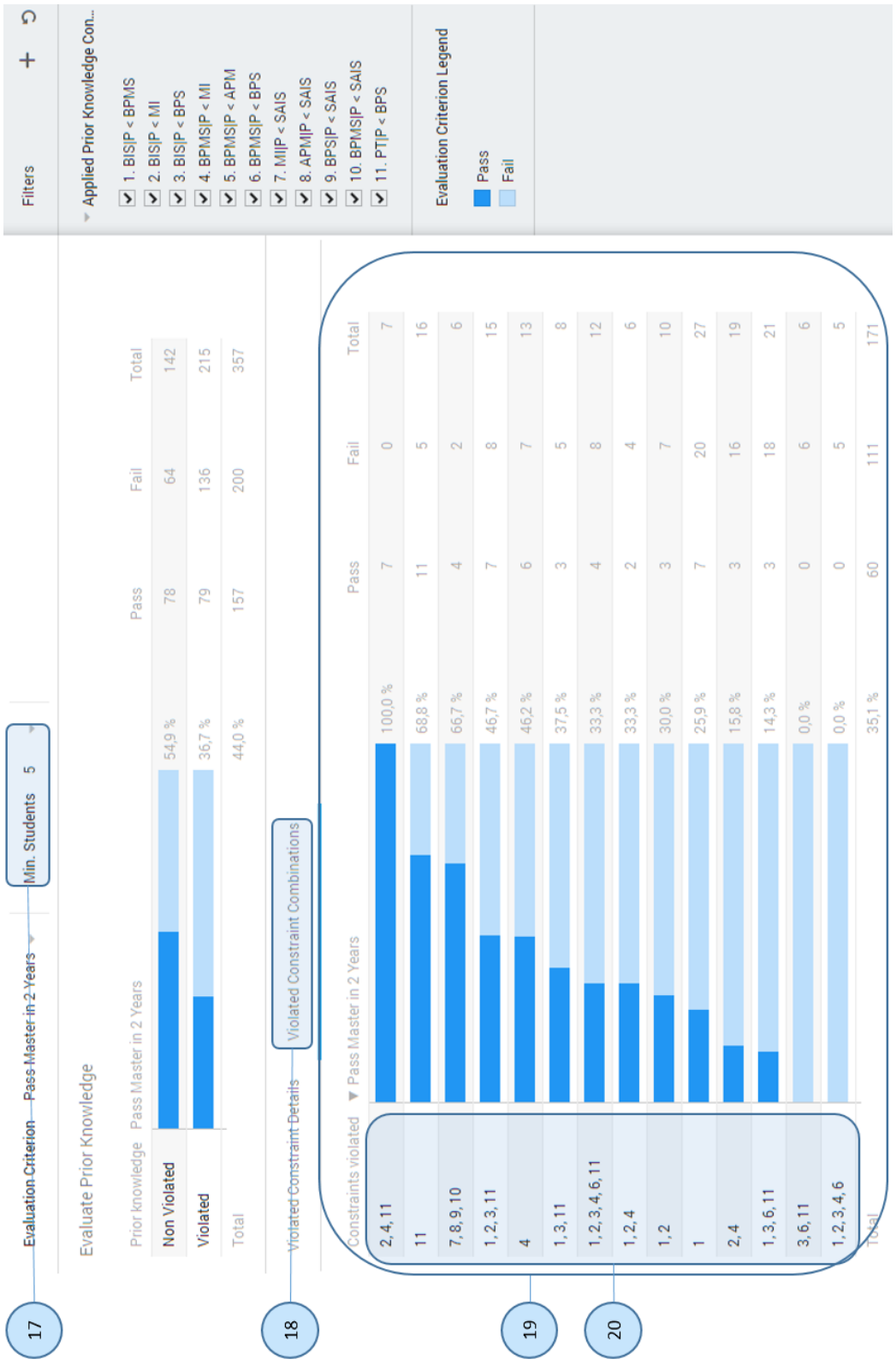


Figure B.2: Overview of the effect of the constraints on the evaluation criterion, only single constraint details

21. The overview dashel which contains all the open cases which are selected in the *Visible Cases* filter.
22. The identifiers for the open cases.
23. These are the traces containing the actions which are relevant and which are already performed (in order). All actions which are performed but are not relevant are not shown in the trace, since this would only clutter the dashel.
24. The current predicted success rate for the evaluation criterion. When there are no similar cases it is impossible to calculate a predicted success rate, therefore a "-" is shown. If the evaluation criteria already evaluated to a success or a fail and cannot change anymore, a ✓ or a ✗ is shown.
25. The number of similar cases on which the success rate from the previous item are based.
26. This blue rectangle shows the selected case. The selected case is shown on the right dashel.
27. These are warning symbols. These are shown when there are no similar cases, such that a success rate can be calculated. These are also shown, when the amount of similar cases is lower than the minimum selected in the *Minimum Similar Cases* filter (located at the same place as in the other visualization).
28. This is the dashel which contains information for the best continuation action.
29. The information which belongs to the current case but is not dependent on which continuation action is chosen. This information contains the case identifier, the actions already performed and the current possibility at a success.
30. The *Visible Cases* filter. It might happen that during a moment in the process there are a lot of open cases. This makes the list in the left dashel too long and the overview becomes cluttered. If a selection is made in this filter, the list becomes clear again. By default, all the cases are selected in this filter.
31. These are the possible continuation actions. Actions which have no influence on the constraints are not shown, since performing one of them would result in the same state. When there is not enough information to calculate statistics for one of the continuation actions, a warning icon is shown in front of the actions. If one of the actions is already performed, and doing it again has no influence anymore on the result, the action is displayed in gray.
32. The predicted success rate if the action in associated category is performed next.
33. The improvement of the predicted success rate compared to the current success rate. This improvement is not relative to the current probability, but just the difference between the two probabilities.
34. The number of similar cases on which the predicted success rate is based.

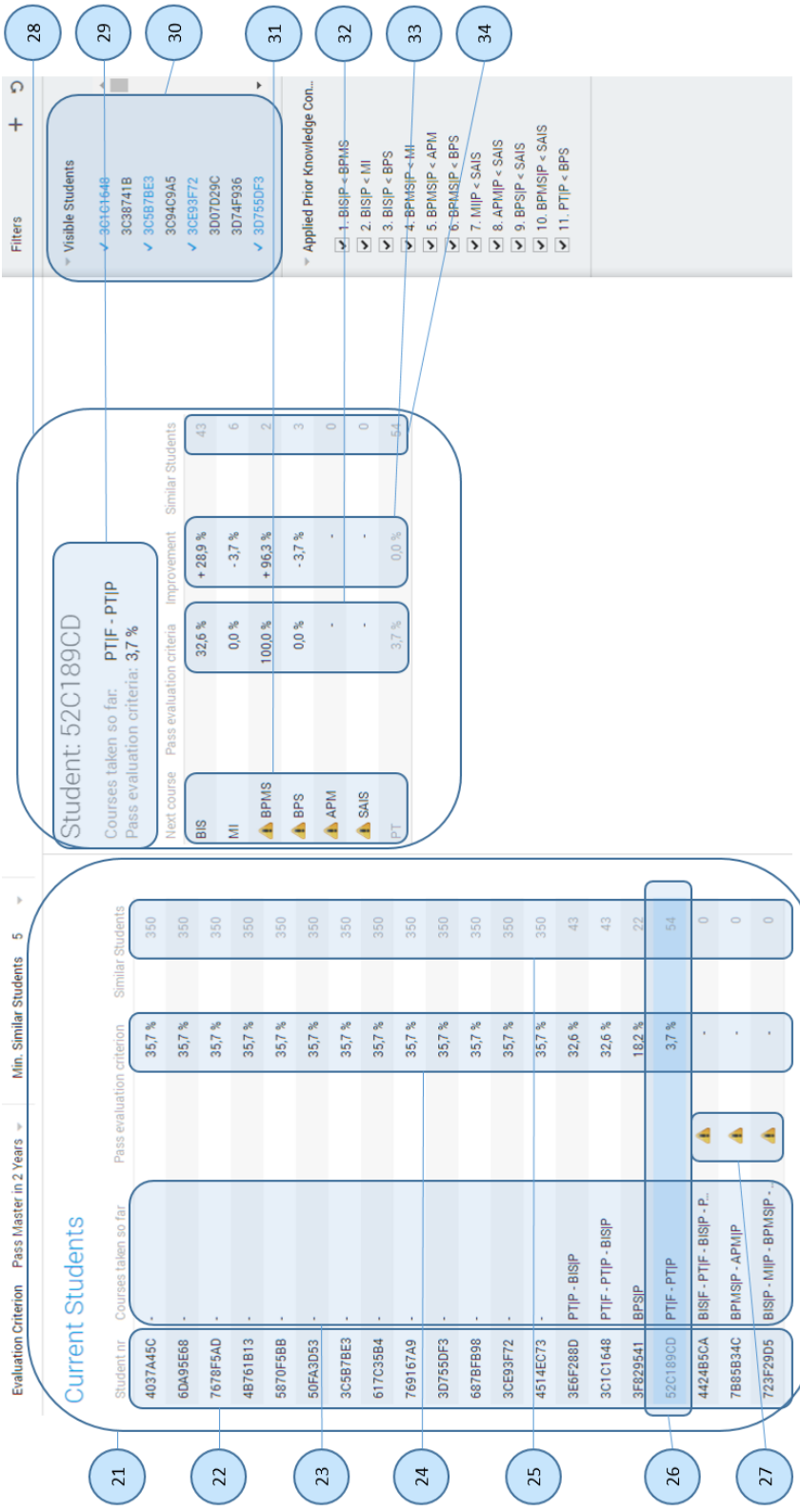


Figure B.3: Overview for suggestions for continuation actions, for a selection of cases

Appendix C

Courses Program

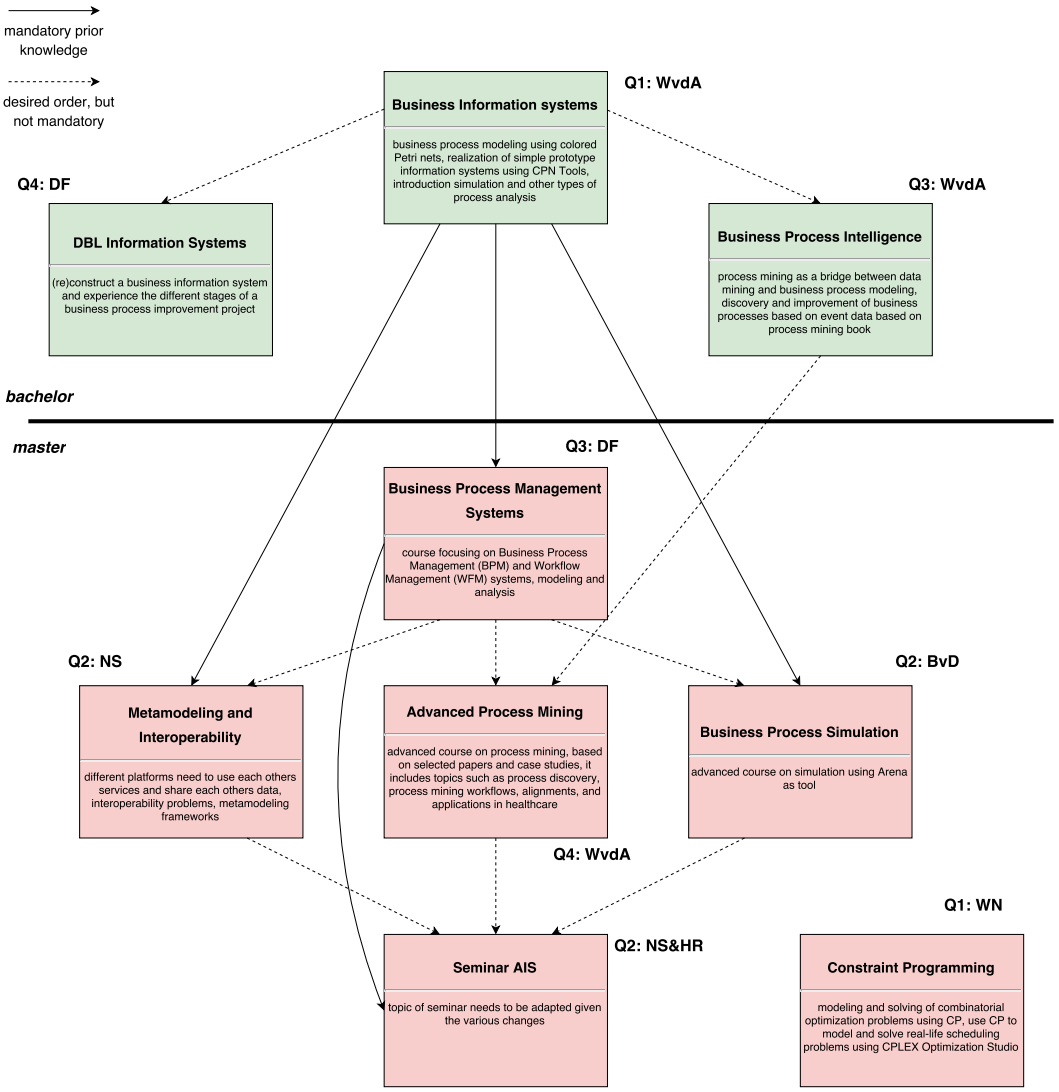


Figure C.1: Course program of courses related to the AIS department

Prior-Knowledge	**Success Rate**	**Pass #**	**Fail #**	**Total #**
Non Violated	73,2 %	104	38	142
Violated	58,6 %	126	89	215
Constraint 1	50,4 %	65	64	129
Constraint 2	56,8 %	54	41	95
Constraint 3	47,8 %	44	48	92
Constraint 4	61,7 %	50	31	81
Constraint 5	-	0	0	0
Constraint 6	45,6 %	31	37	68
Constraint 7	52,9 %	9	8	17
Constraint 8	68,0 %	17	8	25
Constraint 9	62,5 %	10	6	16
Constraint 10	46,2 %	6	7	13
Constraint 11	61,7 %	71	44	115

Table C.1: Results for the *Delay* criterion

Appendix D

Deriving Automata for the Hospital

In this appendix the automata derived from the FLTL formulas in Section 8.2 are shown. Most of the FLTL formula were either of the form that if one activity occurred, another should also occur during the execution, or of the form that one activity should precede another activity. As already briefly mentioned, it was not possible to create an automaton for the constraint "*Each test has at most 3 isolates*". It is possible to create an automaton for each test on its own, but it is not defined how the automaton should look like on intake level. It is possible to create an automaton with two states, the start state and the end state. If one of the tests has more than 3 isolates, there is an arrow from the start to the end state. With this automaton it is not defined when an intake is similar to another intake and therefor the decision is made to not create this automaton. The other automata are shown in the figures below.

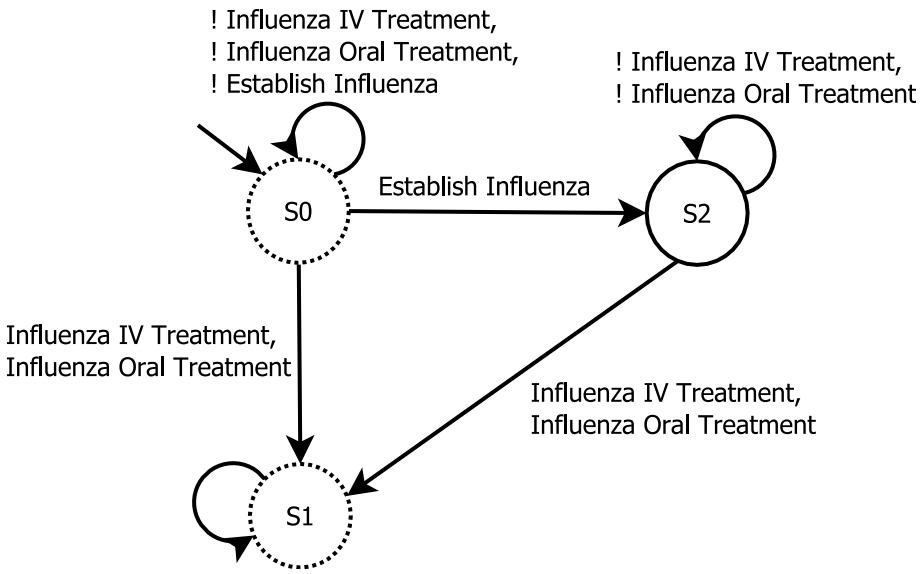
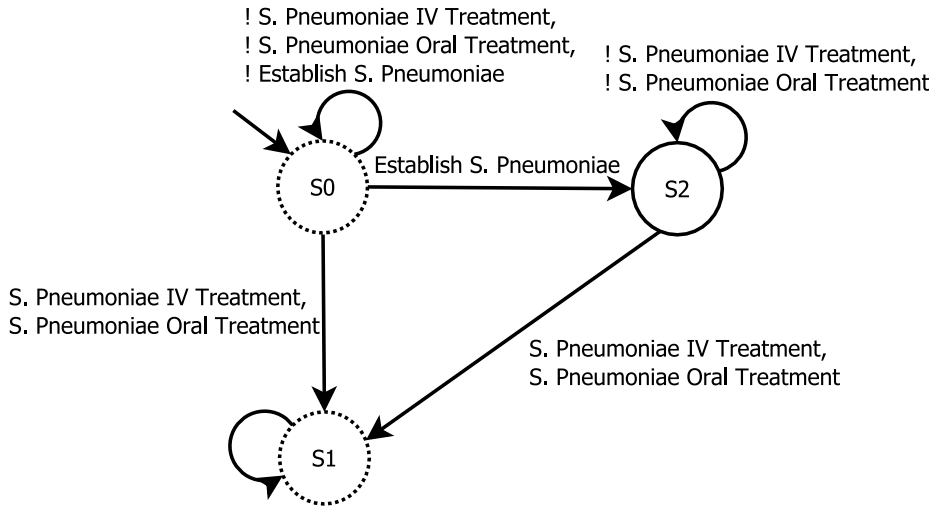
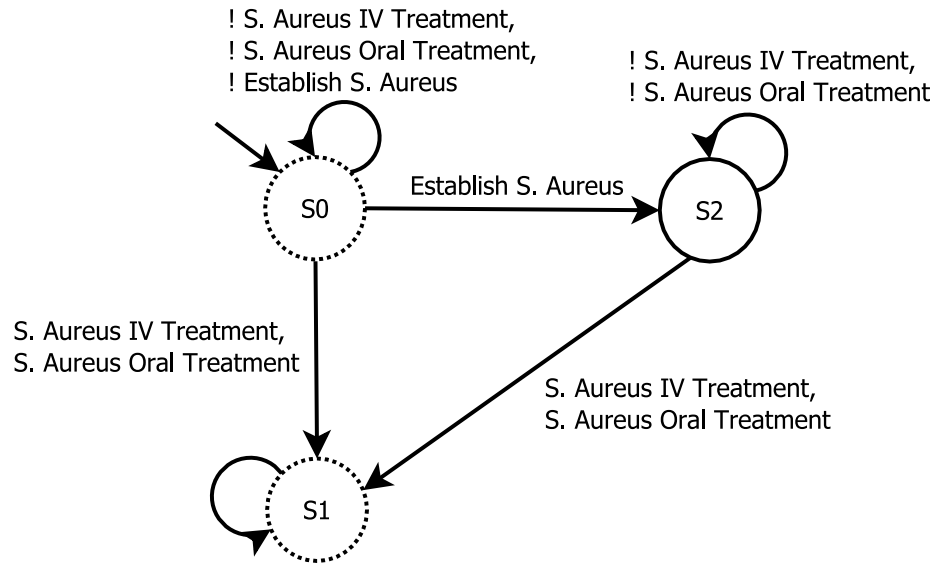


Figure D.1: When Influenza is established, either Oral or IV antibiotics for Influenza should be given

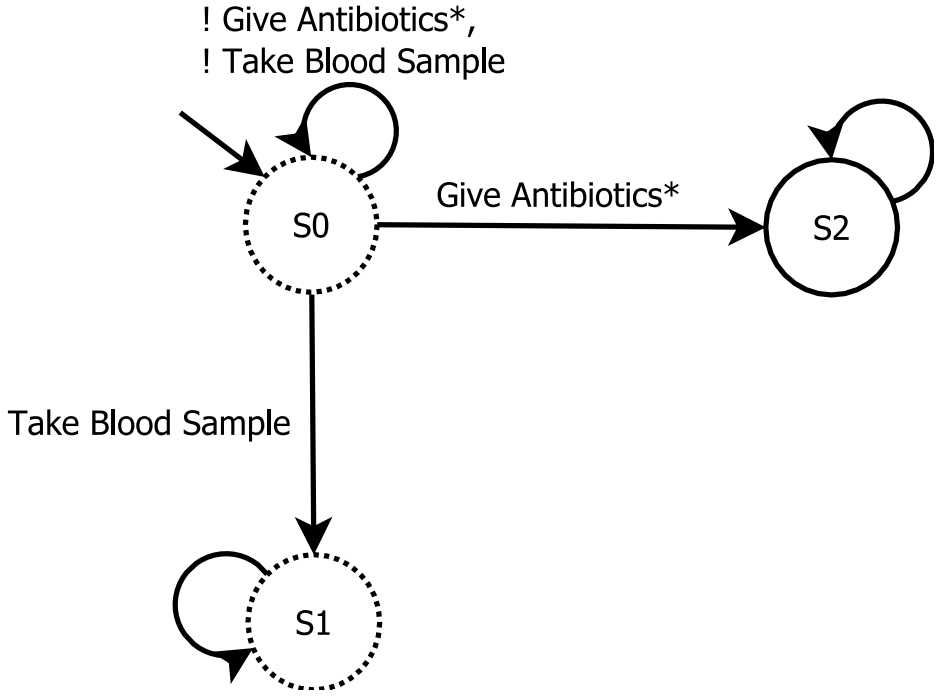


(a) When Streptococcus Pneumoniae is established, either Oral or IV antibiotics for Streptococcus Pneumoniae should be given

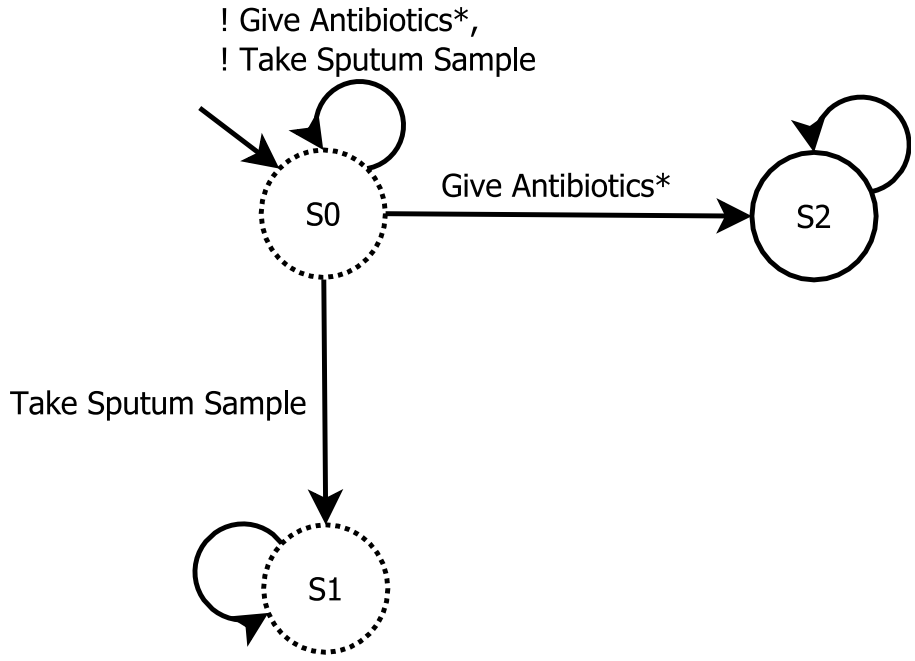


(b) When Staphylococcus Aureus is established, either Oral or IV antibiotics for Staphylococcus Aureus should be given

Figure D.2: Automata for constraints 2 & 3

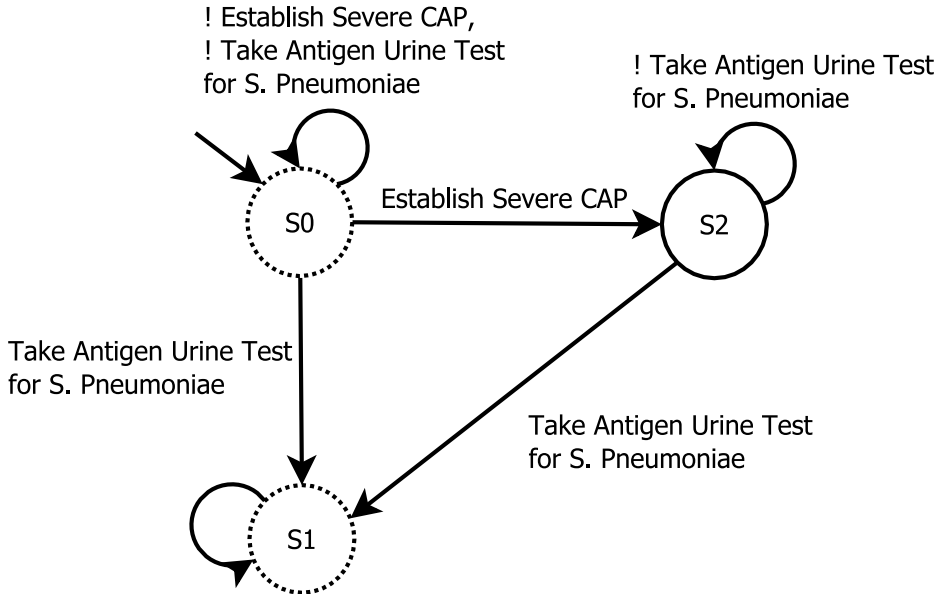


(a) A blood sample should be taken before any antibiotics are give

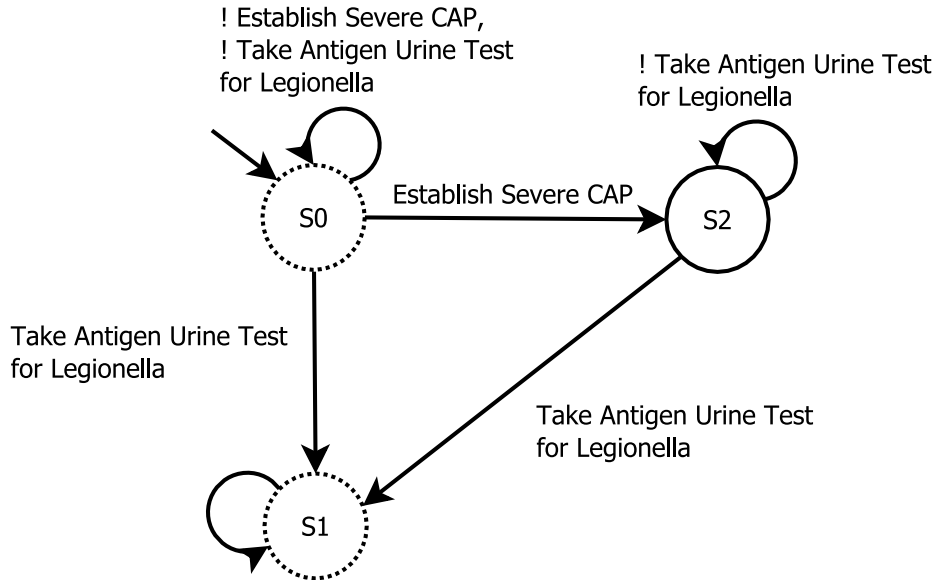


(b) A sputum sample should be taken before any antibiotics are given

Figure D.3: Automata for constraints 4 & 5

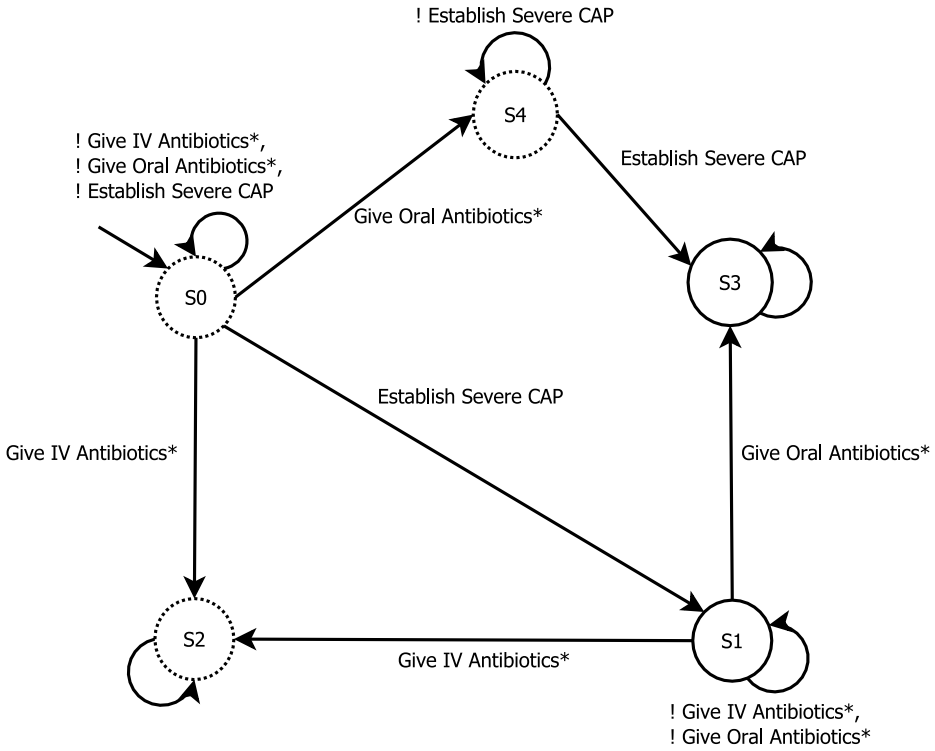


(a) When severe CAP is established, an antigen Urine sample should be taken to test for Streptococcus Pneumoniae

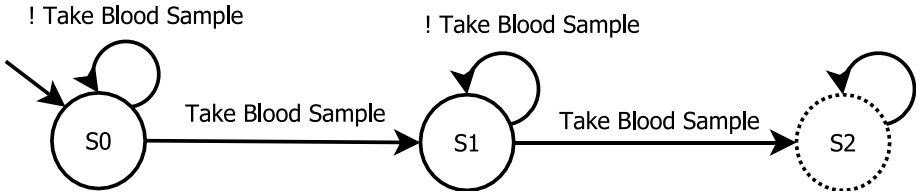


(b) When severe CAP is established, an antigen Urine sample should be taken to test for Legionella

Figure D.4: Automata for constraints 7 & 8



(a) When severe CAP is established, the treatment should start with IV antibiotics



(b) At least two blood tests should be taken per patient

Figure D.5: Automata for constraints 9 & 10