

(Quasi-)Inconsistency Library for Business Rule Management

Masterarbeit

zur Erlangung des Grades eines Master of Science
im Studiengang Wirtschaftsinformatik

vorgelegt von

Matthias Robert Deisen

Erstgutachter: Prof. Dr. Patrick Delfmann

Institut für Wirtschafts- und Verwaltungsinformatik

Zweitgutachter: M. Sc. Carl Corea

Institut für Wirtschafts- und Verwaltungsinformatik

Koblenz, im März 2019

Erklärung

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

Ja Nein

Mit der Einstellung der Arbeit in die Bibliothek bin ich ein-
verstanden.

Der Veröffentlichung dieser Arbeit im Internet stimme ich
zu.

.....
(Ort, Datum) (Matthias Robert Deisen)

Zusammenfassung

Geschäftsregeln sind zu einem wichtigen Instrument geworden, um die Einhaltung der Vorschriften in ihren Geschäftsprozessen zu gewährleisten. Aber die Sammlung dieser Geschäftsregeln kann verschiedene widersprüchliche Elemente haben. Dies kann zu einer Verletzung der zu erreichenden Compliance führen. Diese widersprüchlichen Elemente sind daher eine Art Inkonsistenzen oder Quasi-Inkonsistenzen in der Geschäftsregelbasis. Ziel dieser Arbeit ist es, zu untersuchen, wie diese Quasi-Inkonsistenzen in Geschäftsregeln erkannt und analysiert werden können. Zu diesem Zweck entwickeln wir eine umfassende Bibliothek, die es ermöglicht, Ergebnisse aus dem wissenschaftlichen Bereich der Inkonsistenzmessung auf Geschäftsregel-Formalismen anzuwenden, die tatsächlich in der Praxis verwendet werden.

Abstract

Business rules have become an important tool to warrant compliance at their business processes. But the collection of these business rules can have various conflicting elements. This can lead to a violation of the compliance to be achieved. This conflicting elements are therefore a kind of inconsistencies, or quasi inconsistencies in the business rule base. The target for this thesis is to investigate how those quasi inconsistencies in business rules can be detected and analyzed. To this aim, we develop a comprehensive library which allows to apply results from the scientific field of inconsistency measurement to business rule formalisms that are actually used in practice.

Contents

1	Introduction	1
1.1	Research Questions	2
1.2	Logic of Investigation	2
1.3	Research Method	3
2	Preliminaries	5
2.1	Business Rules	5
2.1.1	Defeasible Logic	5
2.1.2	Declarative Process Model	7
2.1.3	Decision Model and Notation (DMN)	10
2.2	Business Process Model Notation	12
2.3	Inconsistencies	13
2.3.1	Minimal Inconsistent Subsets	13
2.3.2	Minimal Quasi Inconsistent Subsets	19
2.4	Inconsistency Measures	25
3	Implementation	27
3.1	Used Libraries	27
3.2	Finding Paths	29
3.3	Business Rules	30
3.3.1	Defeasible Logic	30
3.3.2	Declarative Process Model	36
3.3.3	DMN	41
3.4	BPMN	49
3.4.1	Graph	51
3.4.2	Checking DECLARE Constraints in BPMN	51

3.4.3	Limitations	57
4	Evaluation	58
4.1	Business Rules	58
4.1.1	Defeasible Logic	58
4.1.2	Declarative Process Model	61
4.1.3	DMN	63
4.2	BPMN with DECLARE	65
5	Conclusion	67
A	Verification Capabilities	74
B	Usage of Library	75
C	BPMN	77
D	Evaluation FCL Solver	79
D.1	Second Configuration	79
D.2	Third Configuration	81
D.3	Fourth Configuration	83

List of Figures

1	The subsumption map of DECLARE templates.[DCMMM17]	15
2	Declare language class diagram	36
3	Transformed \mathcal{C}_1	37
4	Number Line	43
5	Example BPMN model	55
6	Graph for example model	56
7	Runtime statistics for the FCL solver	59
8	MI in FCL test data	60
9	MQI in FCL test data	60
10	Number of MQI in DMN evaluation data set	63
11	Runtime statistics for the DMN solver	64
12	BPMN Hohenheim	78
13	Runtime statistics for the FCL solver (2)	80
14	MI in FCL test data 2	80
15	MQI in FCL test data 4	81
16	Runtime statistics for the FCL solver (3)	82
17	MI in FCL test data 3	82
18	MQI in FCL test data 3	83
19	Runtime statistics for the FCL solver (4)	84
20	MI in FCL test data 4	84
21	MQI in FCL test data 4	85

Chapter 1

Introduction

Business processes describe a sequence of company activities. This sequence along with the semantics of respective activities are mostly predefined by the businesses using various process modelling methods, standards and notations. Here, a central goal for businesses is that the sequence of execution of these business activities has to be compliant with standards, internal policies and current laws. To achieve this, companies can warrant compliance by describing constraints within the business, also known as business rules. These business rules are defined as „compact, atomic, well-formed, declarative statement[s] about an aspect of a business that can be expressed in terms that can be directly related to the business [...]“ [Gra07]. Those „are part of fundamental business practices and policies of the enterprises“ [CA16].

To organize those business rules they are mostly maintained in a central business rule base. But with the complexity of the business and more regulating laws it can be difficult to keep this rule base correct within itself [NPRS10] [Roh05]. This means with increasing complexity of a rule base it is more prone to become inconsistent and therefore cannot be used for its intended purpose of governing compliant processes[CD18a].

This risk of having inconsistencies in a central environment like a business rule base can have significant impacts on the enterprise. Inconsistencies increases the risk to violate policies and laws which can lead to legal affairs.

Thus, it is an important challenge to find those inconsistencies and resolve them in order to reduce this risk for companies. This challenge has recently attracted more interest in research [DCMM13, CD18a, GM18, CD18b].

Furthermore it is interesting how these business rules can be helpful to align the sequence of activities known as business process to a compliant business rule base. This can prevent errors as well as deadlocks in business processes which can be also a high risk for enterprises and therefore can be a big cost factor. This also has become increasing attention of current research [ALR15][GK07].

In this work, we therefore investigate how inconsistencies in business rules can be detected and analyzed. To this aim, we develop a comprehensive library which allows to apply results from the scientific field of inconsistency measurement to business rule formalisms that are actually used in practice.

1.1 Research Questions

The main objective of this thesis is how to find inconsistencies in business rule bases for three different specifications: Formal Contract Language (FCL or Defeasible Logic), DECLARE (or Declarative Process Modell) and Decision Model and Notation (DMN). These three formalisms are common standards in business environments.

In the incremental and collaborative process of organizing business rules, inconsistencies can occur in all of the three business rule languages, for example due to the size and complexity of rule bases, a lack of oversight, inexperienced modelers or different views on the same domain of interest. Furthermore, logical contradictions can occur, which are not an inconsistency per se but can potentially cause inconsistencies. These are called quasi inconsistencies and also need to be found in rule bases.

Another objective is how to check a business process model with business rules given to check the process of its compliance to the business rule base. This is examined on the example of process models in Business Model and Notation (BPMN) specification with DECLARE constraints.

1.2 Logic of Investigation

In Chapter 2 we provide background information on what business rules are and how they are used for different purposes Also it is outlined the possible inconsistencies that can occur within a rule base of a specific implementation of business

rules. Here is important how to find these inconsistencies and how to categorize them.

Chapter 3 focuses on the implementation of finding those inconsistencies within defeasible logic, DECLARE and Decision Tables in DMN (Decision Model and Notation). Furthermore a approach has been made to check the consistency Business Process Models in BPMN (Business Process Model and Notation) with rule base DECLARE.

In Chapter 4 an evaluation of the algorithms, by means of a runtime analysis for different rule bases, is carried out.

Lastly in Chapter 5 the results of the implementation is recapitulated and discussed.

1.3 Research Method

This thesis was conducted by using the design science methodology. Design science research is primarily a „problem solving process“ by implementing a „purposeful artifact“ in a „specified problem domain“ [HMPR04]. In case of this thesis the artifact is the Inconsistency Library with its resolving capabilities in the problem domain of business process modeling and business rule management.

The detection and analysis of inconsistencies in business rule bases in the context of business rule management and business process modeling can be seen as motivated in the introduction as „heretofore unsolved and business problem“ [HMPR04]. Therefore, it seems plausible to follow the approach of solving this problem by applying the methodology of design science [HMPR04, HMPR08, VK04].

This work can be regarded as exaptation research [GH13], since the library was developed by adopting methods from one scientific field, namely inconsistency measurement, to solve problems in another domain.

Various literature ([HMPR04, GH13, VK04]) describes the outcome of design-scientific research as dualistic: to elaborate theories for design and action and to develop usable artifacts. In order to align with this characterization, the thesis was split into two parts.

First, the concepts, definitions and methods from the research on inconsistency measurement were taken up and the plausibility of their application for business rule management and the application of business rules in business process modeling were examined. By adapting and extending the concepts and methods,

this new environment of methods aimed at the detection and analysis of inconsistencies in business rules. Following the environment was used to design and implement a inconsistency library as an applicable artifact. This environment is intended to solve the problem of detecting inconsistencies in business rule management. For this purpose the library was developed by combining the previously adapted and extended concepts and methods.

As suggested in [PTRC07, VPHB16], the developed artifact must be able to demonstrate the applicability of the artifact in order to solve the problem [PTRC07]. For this reason the library was implemented as a prototype and also evaluated, as Venable et al. [VPHB16] exacts for designed artifacts. For the evaluation on how effectively the developed artifact provides a solution to the problem [PTRC07] unit test were set in place to ensure the accuracy and plausibility of the results. Additionally a runtime analysis was carried out to assess the feasibility.

Chapter 2

Preliminaries

In order to comprehend the actions taken to tackle the problem of inconsistency in business rules, we provide background knowledge and definitions for a clear understanding of the problem and parts used for the solution.

2.1 Business Rules

Business Rules are a construct which try to organize a business behaviour and/or its way to how goals of the business are achieved [KEP00]. Business rules are defined as „[...] statements about how the business is done, i.e., about guidelines and restrictions with respect to states and processes in an organization“ [BBG⁺90]. In this thesis three formalisms of business rules are considered: Defeasible logic, the Declarative Process model and Decision Tables in Decision Model and Notation.

All three forms are conform to the definition by Bell et al.[BBG⁺90].

2.1.1 Defeasible Logic

The defeasible logic or Feasible Computational Logic (FCL) was developed by Donald Nute [Nut03] to create a reasoning method that is not monotonic. He proposed a approach which consisted of a closed defeasible theory. Theory in this case means a rule base with FCL rules. He defined a theory as follows:

Definition 1. [Nut03, p.155]

Theory $T = (F, R, C, \prec)$ where

1. F is a set of formulas
2. R is a set of rules
3. C is a set of finite sets of formulas that for every formula ϕ ,
 - $\{\phi, \neg\phi\} \in C$, and
 - for every $S \in C$ and $A \rightarrow \phi$ in R , if $\phi \in S$, then $A \cup (S - \{\phi\}) \in C$
4. \prec is an acyclic binary relation on the non-strict rules in R

Where formula is defined as a literal and a literal is an atomic expression or its corresponding negation. For example ϕ defines a formula, then $\neg\phi$ denotes the complement of ϕ .

Rules consists of formulas and three distinct symbols. Namely \rightarrow for strict rules, \Rightarrow for defeasible rules and \rightsquigarrow for undercutting defeaters.

Definition 2 (Rule definition).

- Strict Rules $A \rightarrow \phi$
- Defeasible rules $A \Rightarrow \phi$
- Undercutting defeaters $A \rightsquigarrow \phi$

All rules are constructed the same way: First a set of formulas A then the rule identifier $\{\rightarrow, \Rightarrow, \rightsquigarrow\}$ and finally the output formula ϕ . The set of formulas A is called an *antedecent* which can not be empty in strict rules and defeaters. When used in a defeasible rules the antedecent can be empty and the rule is then called an *presumption*.

Every rule must not have an exception. Additionally contradicting defeasible rules can have a acyclic relation \prec following named superiority, which specifies which rule is activated when both rules are initially activated. For example:

Example 1.

$$\begin{aligned}
 C_1 = & \\
 & R1 : a \Rightarrow b \\
 & R2 : a \Rightarrow \neg b \\
 & R3 : \gg a \\
 & S1 : R1 \succ R2
 \end{aligned}$$

In this Example 1 $R1$ and $R2$ contradicts each other, but $S1$ states, that in case both $R1$ and $R2$ are activated only $R1$ should be activated, avoiding a situation where an inconsistency can occur because of the contradicting outcome of the rules.

2.1.2 Declarative Process Model

Business Process models are normally described in defined orderly structure in which way a business has to behave, such as processes defined in BPMN 2.2. Whereas the declarative language has a different approach to explain a process via constraints which has to be fulfilled in an instance of the process and must not be violated. In recent time the interest in the declarative process modelling language has increased [CDD19a], [CDD19b], [DAPS09], [BMS16], [DCMMM17].

Definition 3 (Declarative Process Model). [CDD19a] A declarative process model is a tuple $\mathbf{M} = (\mathbf{A}, \mathbf{T}, \mathbf{C})$, where \mathbf{A} is a set of tasks, \mathbf{T} is a set of constraint templates, and \mathbf{C} is the set of actual constraints, which instantiate the template elements in \mathbf{T} with tasks in \mathbf{A} .

This definition is for the DECLARE [Pes08] formalism. Pesic et al. constructed the notation upon Linear temporal logic (LTL). In Table 1 the used LTL syntax and semantic is explained.

Operator	Symbol	Definition
<i>always</i>	$\Box p$	specifies that p holds at every position in the trace
<i>eventually</i>	$\Diamond p$	specifies that p will hold at least once in the trace
<i>next</i>	$\bigcirc p$	specifies that p holds in the next element of the trace
<i>until</i>	pUq	specifies that there is a position where q holds and p holds in all preceding positions in the trace

Table 1 LTL Formula [Pes08, p.122]

In Table 2 the existence relation templates are shown. These templates define information regarding just one event.





Formula	LTL Formula	Graphical Representation
PARTICIPATION(A)	$\diamond A$	
ATMOSTONE(A)	$\neg A \vee (\diamond A \Rightarrow \square(\neg A))$	
INIT(A)	A	
END(A)	$\square \diamond A$	

Table 2 Existence Templates[DCMMM17][CDCDGM18]

The information can be either, if a event occurs at least once (PARTICIPATION), if a event is occurring up to once (ATMOSTONE), if a event starts a sequence (INIT) or if a event ends a sequence (END).

Next up are the relation templates, which define information about two events. They are shown in Table 3 with their LTL formula and graphical representation. The graphical representation can be helpful to understand the individual constraints. The dot in the connection of each representation shows which event activates the formula. If on both ends are dots both events can activate the constraint. For example the RESPONDEDEXISTENCE template is activated through Event *A* whereas the COEXISTENCE Template can be activated through Event *A* or Event *B*.

Formula	LTL Formula	Graphical Representation
RESPONDEDEXISTENCE(A,B)	$\diamond A \Rightarrow \diamond B$	
COEXISTENCE(A,B)	$\diamond A \Leftrightarrow \diamond B$	
RESPONSE(A,B)	$\square(A \Rightarrow \diamond B)$	
PRECEDENCE(A,B)	$(\neg BUA) \vee \square(\neg B)$	
SUCCESSION(A,B)	$\text{RESPONSE}(A, B) \wedge \text{PRECEDENCE}(A, B)$	
ALTERNATERESPONSE(A,B)	$\square(A \Rightarrow \bigcirc(\neg AUB))$	
ALTERNATEPRECEDENCE(A,B)	$\text{PRECEDENCE}(A, B) \wedge \square(B \Rightarrow \bigcirc(\text{PRECEDENCE}(A, B)))$	
ALTERNATESUCCESSION(A,B)	$\text{ALTERNATERESPONSE}(A, B) \wedge \text{ALTERNATEPRECEDENCE}(A, B)$	
CHAINRESPONSE(A,B)	$\square(A \Rightarrow \bigcirc B)$	
CHAINPRECEDENCE(A,B)	$\square(\bigcirc B \Rightarrow A)$	
CHAINSUCCESSION(A,B)	$\square(A \Leftrightarrow \bigcirc B)$	

Table 3 Relation Templates[MMv11][DCMMM17]

The arrow represents the direction within the template. So every relation template is directed except the RESPONDEDEXISTENCE and COEXISTENCE templates. The number of lines represents the type of relation, where the more lines the more restrictive the template is. One line is for the base templates, the two lines restricts that from the activating event the sequence must not contain the activating event again. For example the sequence $[A, A, B]$ checked against the constraint ALTERNATERESPONSE(A, B) shows that the constraint is not satisfied because in the sequence is a sub sequence, where the activating event A occurs twice. Whereas the constraint ALTERNATEPRECEDENCE(A, B) is satisfied because B does not occur twice before the event A is reached in the sequence.

Three lines denotes the CHAIN restriction, which says that the events must be connected directly. For example the constraint CHAINRESPONSE(A, B) defines

that after every event A the next element must be B . If the next element is not B the constraint would not be satisfied.

Formula	LTl formula	Graphical Representation
NOTCOEXISTENCE(A,B)	$\neg(\diamond A \wedge \diamond B)$	
NOTSUCCESSION(A,B)	$\Box(A \Rightarrow \neg(\diamond B))$	
NOTCHAINSUCCESSION(A,B)	$\Box(A \Rightarrow \bigcirc(\neg B))$	

Table 4 Negation Relation Templates[MMv11][DCMMM17]

Furthermore there are negation relation templates in DECLARE which are shown in Table 4. These are the negating templates to their non negation relation templates:

NOTCOEXISTENCE(A, B) defines that if either A or B occurs the other one must not occur in the same instance. NOTSUCCESSION(A, B) defines that if A occurs B must not occur afterwards and if B occurs A must not precede B . For example the instance $[A, B]$ does not satisfy the constraint NOTSUCCESSION(A, B) but $[B, A]$ does, whereas the constraint NOTCOEXISTENCE(A, B) is not satisfied by both instances.

Lastly there is the NOTCHAINSUCCESSION(A, B) constraint which is similar to the NOTSUCCESSION(A, B) constraint with the additional restriction that the events must not occur next to each other. For Example $[A, C, B]$ does satisfy the NOTCHAINSUCCESSION(A, B) constraint because the A and B are not connected directly to each other.

2.1.3 Decision Model and Notation (DMN)

The „Decision Model and Notation“, is a specification proposed by the Object Management Group (OMG) to „[creating] a standardized bridge for the gap between the business decision design and decision implementation“ [dmn18].

This thesis focuses only on a part of the DMN specification namely the Decision Tables. Which represents all input and output variables in a table form.

Example 2 (Decision Table).

	Input			Output
A	Persons Credit Rating from Bureau	Person Credit Card Balance	Person Education Loan Balance	Person Loan Compliance
1	A	< 10000	< 50000	<i>Compliant</i>
2	Not(A)	-	-	<i>Not Compliant</i>
3	-	>= 10000	-	<i>Not Compliant</i>
4	-	-	>= 50000	<i>Not Compliant</i>

Table 5 Decision Table example: Person Loan Compliance[dmn18]

Table 5 is an example for a well defined decision table. The first column represents the Hit Policy and the number of rows. The next three columns display the input variable. Closing with the last column there is the output variable. It has to be noted that the number of input and output variable can differ from table to table.

As variable type are following allowed: Strings, numbers, dates and booleans.

2.1.3.1 Hit Policies

Decision tables must have one hit policy, which defines how rows are matched against the input variables and how the output is constructed. There are following Hit Policies:

- **Unique:** Only one row can be activated
- **Any:** Multiple rows can be activated, the output of each activated row must be the same.
- **First:** Multiple rows can be activated, only the output of the first activated row is returned
- **Rule Order:** Multiple rows can be activated, the output is returned in order of the rows in the table.

- **Collect:** Multiple rows can be activated, the output is returned as a set with the outputs of each activated row

2.1.3.2 Datatypes

In decision tables there are a few datatypes available:

- Numbers
 - $< x$ Value has to be lower than x
 - $> x$ value has to be greater than x
 - $= x$ or x value has to be equal to x
 - $[x..y]$ value has to be between x and y

Additionally the numbers themselves can have a different type: integer, long or double

- Dates with the same comparison functions as numbers
- Booleans, either *true* or *false*
- Strings which can represent just a string or boolean variable which can also be negated with a *not* function

2.2 Business Process Model Notation

The Business Process Model and Notation (BPMN) is a notation which was proposed by the Object Management Group¹ and implemented as a ISO standard in 2013 [OMG13].

The standard defines a way to model business processes so that most business users can understand its semantic and syntax [OMG13, p.1].

Because BPMN is too extensive to describe in detail and mostly common knowledge in the business modelling domain it is in this thesis mostly omitted.

In this thesis only part of the notation is needed. Namely events, tasks, sequence flows and message flows which are defined as follows:

¹<https://www.omg.org>

Definition 4 (Vertex and Edge Definition in BPMN). [OMG13, p.26ff]

- **Event:** An Event is something that „happens“ during the course of a Process. These Events affect the flow of the model and usually have a cause (trigger) or an impact (result).
- **Activity:** An Activity is a generic term for work that company performs in a Process. An Activity can be atomic or non-atomic (compound).
- **Task:** A Task is an atomic Activity that is included within a Process. A Task is used when the work in the Process is not broken down to a finer level of Process detail.
- **Sequence Flow:** A Sequence Flow is used to show the order that Activities will be performed in a Process.
- **Message Flow:** A Message Flow is used to show the flow of Messages between two Participants that are prepared to send and receive them.

These vertices and edges are needed for building up sequences of activities which can be executed by the enterprise which implemented the process model. An example of a Business process model in BPMN is Figure 12.

2.3 Inconsistencies

Inconsistencies in the domain of business rules are business rule bases which contains logical conclusions which contradict each other.

2.3.1 Minimal Inconsistent Subsets

Following the definition of a minimal inconsistent subset from Hunter:

Definition 5 (Minimal Inconsistent Subset [HK⁺08]).

$$MI(K) = \{C \subseteq K \mid C \vdash \perp \wedge \forall C' \subset C, C' \not\vdash \perp\}$$

This definition means that for every belief case K a minimal inconsistent subset C is a subset of K where the MI is inconsistent and every genuine subset of the MI is consistent.

2.3.1.1 Inconsistencies in FCL

As outlined before the superiority in Example 1 is important to avoid an inconsistency. In this example without $S1$, $R3$ would enable both $R1$ and $R2$ but which contradict each other. So the Example 1 without $S1$ is an MI because eliminating every other rule would make it not inconsistent anymore.

Example 3 (MI example in FCL).

$$\begin{aligned} \mathcal{C}_1 = \\ R1 : a \Rightarrow b \\ R2 : a \Rightarrow \neg b \\ R3 : \gg a \end{aligned}$$

Example 3 shows a MI in defeasible logic derived from Example 1

2.3.1.2 Inconsistencies in DECLARE

As an example we have following constraint set as an belief case K according to the definition in 2.3.1.

Example 4. MI in DECLARE

$$\mathcal{C}_1 = \{\text{SUCCESSION}(a, b), \quad \text{NOTSUCCESSION}(a, b), \quad \text{PARTICIPATION}(a)\}$$

The MI in this case would be K . That is because of the rules $\text{SUCCESSION}(a,b)$ and $\text{NOTSUCCESSION}(a,b)$ are contradicting each other and are activated by the rule $\text{PARTICIPATION}(a)$. So every constraint in K is part in MI and deleting any constraint would make it not inconsistent anymore.

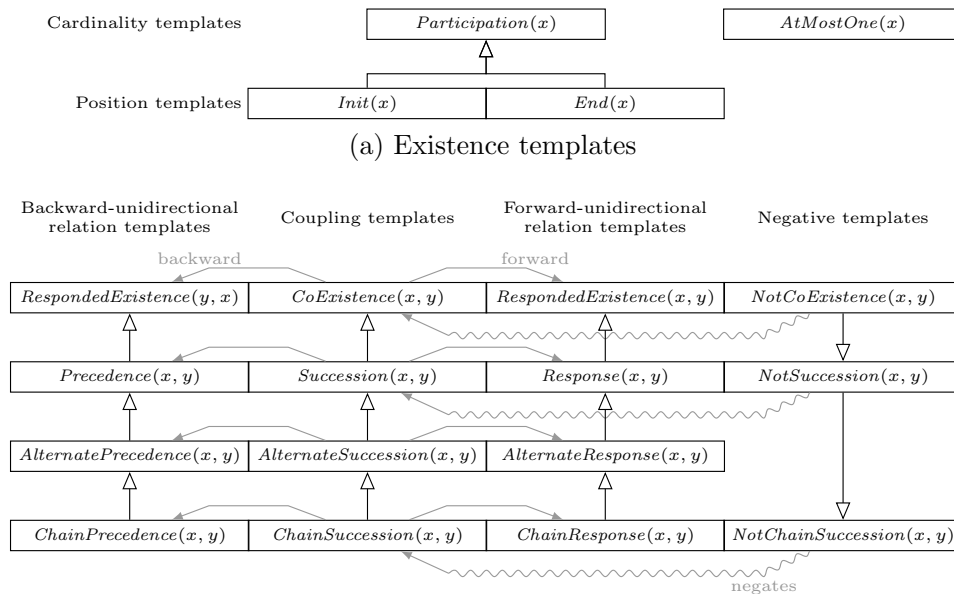


Figure 1 The subsumption map of DECLARE templates.[DCMMM17]

In Figure 1 every possible contradicting constraint is shown. This subsumption map is in the style of an UML Class Diagram², where the arrows show the inheritance of properties as well as the unidirectional association of the coupling templates.

For every NOT relation template are specific constraints that contradict them. For example, the NOTCHAINSUCCESSION can only be contradicted by the CHAIN templates (in the figure in the same row). Whereas the NOTCOEXISTENCE template can be contradicted by every over non negation relation template (in the figure all columns to the left of the negation templates). This is because of inheritance of properties within the various types of relation templates.

In addition to two contradicting constraints there has to be a existence template which activates the constraints. All existence templates can be activate the constraint except the ATMOSTONE template which does not specify if an event has to be used.

²<https://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF>

2.3.1.3 Decision Tables

There can be multiple ways in decision tables to violate the consistency. Mainly it can be said that any decision table is consistent in the term, that there is no MI because the input for the tables must be given in order to determine which rows are activated therefore if there are any contradicting rows which are a MI.

Hit Policies

Furthermore there is also a possibility to violate Hit Policies in Decision Tables. Two of the hit policies listed in can be violated:

- **Unique:** If more than one row is activated
- **Any:** More rows are activated, but the outputs of the rows are not the same.

The other hit policies do not specify that the output values have to be the same. For most they only define in which order the outputs must be returned, which does not influence any possible MI.

One table

One decision table can be inconsistent if the input activates two rows within a table which output is contradicting each other.

Example 5 (Inconsistent Decision Table). Note the following table with the input set I :

- Credit Rating = A
- Credit Card Balance = 10000
- Loan Balance = 5000

	Input			Output
C	Credit Rating	Credit Card Balance	Loan Balance	Loan Compliance
1	<i>A</i>	≤ 10000	< 50000	<i>Compliant</i>
2	<i>Not(A)</i>	-	-	<i>Not Compliant</i>
3	-	≥ 10000	-	<i>Not Compliant</i>
4	-	-	≥ 40000	<i>Not Compliant</i>

Table 6 Decision Table example: Person Loan Compliance [dmn18]

In Example 5 the given input variables activates multiple rows in the Table 6. Namely Row 1 and Row 3. The output of those two rows contradicts each other. Resulting in a MI, which consists of Row 1, Row 3 and the Input I .

Multiple tables

Inconsistencies can also occur with multiple decision tables involved. This is mainly the case if a table is dependent on an other table.

Example 6 (Inconsistencies across two Decision Tables).

Consider those two decision tables, where Table 7 is dependent on Table 8. This is because the output *CreditRating* of Table 8 refers to the input in Table 7.

Additionally consider the following Input set I for the decision tables:

- Credit History = *False*
- Condominium = *True*
- Credit Card Balance = 10000
- Loan Balance = 5000

	Input			Output
C	Credit Rating	Credit Card Balance	Loan Balance	Loan Compliance
1	<i>A</i>	≤ 10000	≤ 50000	<i>Compliant</i>
2	<i>B</i>	-	-	<i>Not Compliant</i>
3	<i>C</i>	≥ 10000	-	<i>Not Compliant</i>
4	-	-	≥ 50000	<i>Not Compliant</i>

Table 7 Decision Table example

	Input		Output
C	Credit History	Condominium	Credit Rating
1	<i>False</i>	<i>True</i>	<i>A</i>
2	-	<i>True</i>	<i>B</i>
3	<i>False</i>	<i>False</i>	<i>C</i>
4	<i>False</i>	-	<i>D</i>

Table 8 Decision Table example

With the input I is each individual table not inconsistent. In Table 7 no row is activated due to the fact that the variable *CreditRating* is not available. Whereas in Table 8 Row 1 and Row 2 are activated with the input variables *CreditHistory* and *Condominium* from the input I . But the output of the two rows are not contradicting each other making the table per se not inconsistent.

Combining these two tables however, results in an inconsistency. That is due to the fact that as outlined before in Table 8 two rows are activated (Row 1 and Row 2) with the output $CreditRating = \{A, B\}$. This output as well as the input

set I activates Row 1 and Row 2 in Table 7. These two rows have contradicting outputs, resulting in an inconsistency with the corresponding MI.

MI =
 I ,
 Row 1 @ Table 8,
 Row 2 @ Table 8,
 Row 1 @ Table 7,
 Row 2 @ Table 7

2.3.2 Minimal Quasi Inconsistent Subsets

Corea et al. [CDD19a] introduces the new approach of minimal quasi inconsistent subset, beginning with this assumption

Lemma 1. [CDD19a] With the current definition of minimal inconsistent subsets, it is not possible to detect inconsistencies in [...] constraint sets C [...]. The proof results from the definition[...] constraint sets C [...] and minimal inconsistent subsets [...], as due to ex falso quodlibet, no reasoning about inconsistent subsets is possible without knowledge of activations.

This lemma was applied to declarative constraint minimal inconsistent subsets, but can be applied to every business rule formalism in this thesis. First to introduce the concept of quasi inconsistencies, the activation of a constraint has to be defined:

Definition 6 (Individual Constraint Activation). [CDD19a] A set of activations A activates an individual constraint $c : a \Rightarrow \varphi$ iff $a \in A$.

In this definition A stands for an activation set, which activates the constraint c with the input a and the output φ . This means that the activation set „activates“ a and through the constraint c , the outcome φ is also activated.

Definition 7 (Constraint Set activation). [CDD19a]
 A set of activations A activates a set of constraints C iff $\forall c \in C : A \cup \{out(c) | c \in C\}$ activates c .

Minimal quasi inconsistent subset or MQI are an derivation of MI defined in 2.3.1. So every MI is also a MQI but every MQI must not be an MI. More general the definition of quasi inconsistent subsets is the following

Definition 8 (Quasi Inconsistent Subset). [CDD19a]

For a constraint set C , the set of quasi inconsistent subsets QI is defined as a set of pairs (A, C) , s.t.

1. $C \subseteq K$
2. A activates C
3. $A \cup C \models \perp$

But in order to have an minimal quasi inconsistent subset there must be applied further restrictions:

Definition 9 (Minimal Quasi Inconsistent Subset). [CDD19a]

For a constraint set C , the set of minimal quasi inconsistent subsets MQI is defined as set of pairs $t = (A, C)$, s.t.

1. t is a quasi inconsistent subset in K
2. $\forall t' \subset t \not\models \perp$

This definition defines the MQI is basically similar to the definition of a QI with the additional requirement that every real subset of the MQI is not inconsistent anymore. That means that removing any element out of the MQI will result in the subset of the MQI not being inconsistent anymore.

Corea et al. formulated the Minimal Quasi Inconsistent Subset definition to describe MQI in DECLARE, but it can be applied to every business rule form considered in this thesis.

Since the definition of MQI and MI both involve conflicting elements, a theorem can be derived from them:

Theorem 1 (Inconsistency Inheritance).

Every MI is also a MQI if the MI does not contain a contradiction in form of two facts contradicting each other

2.3.2.1 Quasi Inconsistencies in FCL

Example 7 is modification of Example 3 which omits $R3$ which was the fact that activates $R1$ and $R2$.

But $R1$ and $R2$ would still contradict each other but without the activation. Although $R1$ and $R2$ share the same antecedent, so that it still can be said, that this is a form of inconsistency. Due to the reason that it fails the requirements of an MI described in 2.3.1 it is considered as an MQI.

Example 7 (MQI example in FCL).

$$\begin{aligned} \mathcal{C}_1 = \\ R1 : a \Rightarrow b \\ R2 : a \Rightarrow \neg b \end{aligned}$$

Example 8 shows a more complex example of an MQI in FCL. The contradicting rules $R1$ and $R2$ share most of their antecedent a and b , but $R1$ has an additional literal as antecedent namely k which would make the $R1$ and $R2$ not an MQI anymore. But due to the fact $R6$ this requirement for $R1$ is satisfied.

Example 8 (Quasi inconsistency in FCL).

$$\begin{aligned} R1 : a, f, k \rightarrow c \\ R2 : h, d \rightarrow \neg c \\ R3 : b \rightarrow d \\ R4 : b \rightarrow f \\ R5 : a \Rightarrow h \\ R6 : \gg k \end{aligned}$$

Making $R1$ and $R2$ quasi inconsistent through the reason that they now have the same activation set $A = \{a, b\}$

2.3.2.2 Quasi Inconsistencies in Declare

Quasi inconsistencies in DECLARE follows the definition of given in Definition 9 (Minimal Quasi Inconsistent Subset).

The Example 9 is an abbreviation of Example 4 by eliminating the PARTICIPATION(a) constraint. Leaving the constraint set with containing only two rules which after Figure 1 contradict each other.

Example 9 (MQI in Declare).

$$\mathcal{C} = \{\text{SUCCESSION}(a, b), \quad \text{NOTSUCCESSION}(a, b)\}$$

But due to the reason that the constraints are only viewed in their constraint set and this set does not contain a existence constraint which activates the rule it is not an inconsistency in the original sense.

Nonetheless the two constraint when activated are still contradicting. Consequential with an activation set the constraint set is considered a quasi inconsistency. In this case the activation set can either be the constraint PARTICIPATION(a) or the constraint PARTICIPATION(b).

2.3.2.3 Quasi Inconsistencies in Decision tables

MQI in decision tables can be found in contrast to MI without the input. Instead the objective for MQI in Decision Tables is to find rows which contradict each other and furthermore can be activated through one input set.

Hit Policies

MQI for the hit policies is the same as in 2.3.1.3. The Unique hit policy can be violated if there is an input I , which activates more than one row. The MQI in this case would be the two rows which are activated through I .

The Any hit policy can be violated if there is an input I , which activates at least two rows which do not have the same output. The MQI in this case would be the two rows which are activated through I .

One table

An MQI in an decision table is similar to an MI (2.3.1.3), with the exception that no input is needed.

According to the definition of Minimal Quasi Inconsistent Subset, a subset t of the belief case K is needed. In case of one decision table this subset t consists of two rows of the decision table, which outputs are contradicting each other. The

activation set A is an input for which the two rows in t are activated.

Given the Example 5 there are in total 3 contradicting output combinations. Row 1 has a contradicting output to Row 2-4. But after Minimal Quasi Inconsistent Subset there must be an activation set, which activates both contradicting outputs.

In Example 5 only two out of three contradicting output combinations have individual activation sets which activates them:

- Row 1 and Row 3:
 - $CreditRating = A$
 - $CreditCardBalance = 10000$
 - $CreditCardBalance = (< 50000)$
- Row 1 and Row 4:
 - $CreditRating = A$
 - $CreditCardBalance = (<= 10000)$
 - $LoanBalance = (40000 <= x < 50000)$

For the Row 1 and Row 2 do have contradicting output, but the $CreditRating$ input for the two rows does not overlap and therefore those rows can never be activated through the same input.

Multiple tables

Minimal quasi inconsistencies can also occur in multiple tables. The corresponding activation set is an input set such as in MQI for one table.

MQI can only occur in multiple decision tables if they are dependent on each other. This means that there can not be an quasi inconsistency if the decision tables do not share information among themselves.

According to the definition of Minimal Quasi Inconsistent Subset, a subset t of the belief case K is needed. In case of multiple decision tables this subset t consists of rows of different decision tables, the outputs of two rows are contradicting each other. The activation set A is an input for which the rows in t are all activated.

Following is an example how a MQI in DMN is constructed over two decision tables.

Example 10 (MQI Example for two depending decision tables).

As shown in 2.3.1.3 the two individual tables do not have any inconsistency, but also no quasi inconsistency. This is because for every input there are no two activated rows with a contradicting output.

But Table 7 is dependent on Table 8 and when combined there are input sets which cause inconsistencies:

- $I_1 =$
 - *CreditHistory* = *False*
 - *Condominium* = *True*
 - *CreditCardBalance* = (≤ 10000)
 - *LoanBalance* = (< 50000)

- $I_2 =$
 - *CreditHistory* = *False*
 - *Condominium* = *True*
 - *CreditCardBalance* = (≤ 10000)
 - *LoanBalance* = 50000

These inputs cause the following MQI:

$$\begin{aligned} \text{MQI}_1 = & \\ & \{I_1, \\ & \text{Row 1 @ Table 8,} \\ & \text{Row 2 @ Table 8,} \\ & \text{Row 1 @ Table 7,} \\ & \text{Row 2 @ Table 7}\} \end{aligned}$$

$$\text{MQI}_2 = \{I_2, \text{Row 1 @ Table 8, Row 4 @ Table 8, Row 1 @ Table 7, Row 4 @ Table 7}\}$$

2.4 Inconsistency Measures

For better overview of inconsistencies in business rule bases measures have been developed [HK⁺08, Thi16, Thi18, GM18]. The measure will help analyzing and rate a knowledge base [NCD19]. There are various measures which focus on different aspects of the inconsistencies in the rule bases.

Corea et al. [CDD19a] takes an approach of measures which are an abbreviation of Hunter et al. [HK⁺08], but in the context of quasi inconsistencies.

Nonetheless the measures are applicable to both MI and MQI. It has to be noted that the measures for MQI will mostly be higher than the for MI because of their definition and Theorem 1.

All following definition are taken from Corea et al. [CDD19a].

Definition 10 (MQI-inconsistency measure).

Define the *MQI-inconsistency measure* via

$$\mathcal{I}_{\text{MI}}^Q(\mathcal{C}) = |\text{MQI}(\mathcal{C})|$$

This measure counts the number of minimal quasi inconsistent subsets in \mathcal{C} .

Definition 11 (MQI^c-inconsistency measure). Define the *MQI^c-inconsistency measure* via

$$\mathcal{I}_{\text{MI}^c}^Q(\mathcal{C}) = \sum_{M \in \text{MQI}(\mathcal{C})} \frac{1}{|M^c|}$$

This measures aggregates the number of minimal quasi inconsistent subsets, normalized by the respective size.

Definition 12 (Quasi problematic-inconsistency measure).

Define the *quasi problematic-inconsistency measure* via

$$\mathcal{I}_p^Q(\mathcal{C}) = \left| \bigcup_{M \in \text{MQI}(\mathcal{C})} M^C \right|$$

This measure counts the distinct number of constraints participating in any minimal quasi inconsistent subset.

Definition 13 (Quasi mv-inconsistency measure).

Define the *quasi mv-inconsistency measure* via

$$\mathcal{I}_{mv}^Q(\mathcal{C}) = \frac{|\bigcup_{M \in \text{MQI}(\mathcal{C})} \mathcal{A}(M^C)|}{|\mathcal{A}(\mathcal{C})|}$$

This measure expresses the ratio of tasks involved in any minimal quasi inconsistent subsets.

Definition 14 (Cardinality-Based Culpability Measure). Define the *cardinality based culpability measure* $\mathcal{C}_\#$ via

$$\mathcal{C}_\#(\mathcal{C}, \alpha) = |\{M \in \text{MQI}(\mathcal{C}) \mid \alpha \in M^C\}|$$

This measure counts the number of minimal quasi-inconsistent subsets that a constraint α appears in.

Definition 15 (Normalized Cardinality-Based Culpability Measure). Define the *normalized cardinality based culpability measure* \mathcal{C}_c via

$$\mathcal{C}_c(\mathcal{C}, \alpha) = \sum_{M \in \text{MQI}(\mathcal{C}) \text{ s.t. } \alpha \in M^C} \frac{1}{|M^C|}.$$

This measure counts the number of minimal quasi-inconsistent subsets that a constraint α belongs to, normalized by the cardinalities of the respective subsets. All the above measures helps to identify to which grade an belief case is inconsistent.

Chapter 3

Implementation

For the implementation Java¹ as the main programming language was used. That is because Java is a widely supported platform² with a wide variety of accessible and free additional libraries.

3.1 Used Libraries

In addition to Java different open source libraries were used and are listed below.

- SPINdle: logic reasoner that can be used to compute the consequence of defeasible logic theories in an efficient manner.³
- Google's guava project: set of core libraries that includes new collection types (such as multimap and multiset), immutable collections, a graph library, functional types, an in-memory cache, and APIs/utilities for concurrency, I/O, hashing, primitives, reflection, string processing, and more⁴
- Camunda bpmn model API: The camunda BPMN model API is a simple, lightweight Java library for parsing, creating and editing BPMN 2.0 XML files.⁵

¹<https://www.java.com/de/>

²<https://www.java.com/de/download/manual.jsp>

³<http://spindle.data61.csiro.au/spindle/index.html>

⁴<https://github.com/google/guava/>

⁵<https://github.com/camunda/camunda-bpmn-model>

- Camunda dmn engine: Lightweight Execution Engine for DMN (Decision Model and Notation).⁶
- Junit: serves as a foundation for launching testing frameworks on the Java Virtual Machine (JVM)⁷
- dmn-generator⁸: Library to generate random DMNs and tests the performance of the solving engine
- ANTLR: (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files.⁹
- JAXB: The Java Architecture for XML Binding (JAXB) provides an API and tools that automate the mapping between XML documents and Java objects.¹⁰

The usage for each of the libraries is documented in Table 9.

	FCL	DECLARE	DMN	BPMN
SPINdle	✓			
Guava		✓		✓
BPMN API				✓
DMN Engine			✓	
Junit	✓	✓	✓	✓
DMN generator			✓	
ANTLR		✓		
JAXB			✓	✓

Table 9 Usage of libraries

⁶<https://github.com/camunda/camunda-engine-dmn>

⁷<https://junit.org/junit5/>

⁸<https://gitlab.uni-koblenz.de/fg-bks/dmn-fcl-converter/wikis/Performance/Performance-tester>

⁹<https://www.antlr.org/>

¹⁰<https://github.com/eclipse-ee4j/jaxb-ri>

3.2 Finding Paths

For implementing the solver of the inconsistencies mainly a graph based approach was made. For most of the formalisms there is a logically transfer method into a graph. As a graph datatype the Google guava project was used¹¹.

The need was to compute paths within this graph, because the graph was implemented generic, a generic algorithm can be implemented. This path finding algorithm is based upon the Depth-first search [CLRS09, p.603ff]. Additionally the paths found by the Depth-first search algorithm are stored and returned as an output to the *findPaths* function, which is shown in Algorithm 1. The function

Algorithm 1: findPaths function

Given : Graph $G = (N, E)$
Input : Start node $n \in N$
Output: Paths $P = \{P_1, \dots, P_n\}$

- 1 $P \leftarrow \emptyset$
- 2 $p_n \leftarrow \{n\}$
- 3 $succ \leftarrow successors(n)$
- 4 **foreach** $s \in succ$ **do**
- 5 $P_s \leftarrow \emptyset$
- 6 $p_{s_n} \leftarrow p_n \cup s$
- 7 **foreach** $p \in findPaths(s)$ **do**
- 8 $P_s \leftarrow P_s + \{p_{s_n} \cup p\}$
- 9 $P \leftarrow P + P_s$

takes as input the start node n and queries the following nodes in the graph with the *successors()* function. Now for each successor s a base path with the starting node and the successors is created and stored in the variable p_{s_n} (Line 6).

At this point a recursive function call is executed with the successor s as the new starting node. For each of the returned path p the base path from the beginning is added, so that in P_s all paths from the starting node n with its successor s are stored. Finally all paths are combined and stored into the output variable P .

¹¹<https://github.com/google/guava/wiki>

3.3 Business Rules

The main objective of this thesis is to implement algorithms which find inconsistencies in form of MI (defined in Minimal Inconsistent Subsets) and all quasi inconsistencies MQI (defined in Minimal Quasi Inconsistent Subset).

3.3.1 Defeasible Logic

The syntax and semantics of defeasible logic is already implemented in the SPINdle library (3.1) which also has a working reasoning engine for FCL. But SPINdle can not handle inconsistencies within theories. In order to identify those the SPINdle library has to be extended.

3.3.1.1 Finding inconsistencies

The first step to find inconsistencies within a theory is to find all contradicting rules. In other words the rule where negated outcome of an rule exists also as an outcome, e.g. a and $\neg a$. If there are no contradicting rules there are also no inconsistencies.

If there are rule pairs which are contracting, for each rule the corresponding paths are computed with the help of the reasoner (Algorithm 2, Line 5). For finding the paths the antecedents are stored and checked if there are any other rules with the literal of the antecedents as outcome. This is recursively done until there are no rules with the literal as outcome there or the rule found is a fact.

For each found path the reasoner was used to determine if the path is valid (the first rule is activated).

If both rules have valid paths, all of the paths found are combined and stored as an MI (Line 8-10).

Algorithm 2: Computation of Minimal Inconsistent Subsets

Output: $\text{MQI}(\mathbf{C})$
Input : Theory $T = (K, L)$

```

1  $mis \leftarrow \emptyset$ 
2 for  $i$  from 1 to  $|K|$  do
3    $P_i \leftarrow \emptyset$ 
4 foreach  $\alpha \in K$  do
5    $P_\alpha = \{M_1, \dots, M_m \mid M \subseteq K \wedge M \vdash \alpha\}$ 
6 foreach  $\alpha \in K$  do
7   if  $\exists \alpha_y$  s.t.  $\beta_{h_{\alpha_y}} == \neg \beta_{h_\alpha}$  then
8     foreach  $P' \in P_\alpha$  do
9       foreach  $P'' \in P_{\alpha_y}$  do
10         $mis = mis + \{\{P'\} \cup \{P''\}\}$ 

```

Example 11 (Finding MI in a FCL Theory).

Consider following FCL theory T :

```

 $r_1$ : >> person(John)
 $r_2$ : >> creditEligible(John)
 $r_3$ : person(X)  $\Rightarrow$  contractuallyCapable(X)
 $r_4$ : >> mentalCondition(John)
 $r_5$ : mentalCondition(X)  $\Rightarrow$   $\neg$  contractuallyCapable(X)
 $r_6$ : mentalCondition(X)  $\Rightarrow$   $\neg$  creditEligible(X)
 $r_7$ :  $\neg$  creditEligible(X)  $\Rightarrow$   $\neg$  grantCredit(X)
 $r_8$ :  $\neg$  person(X)  $\rightarrow$   $\neg$  grantCredit(X)

```

Now for each rule r in T the paths are computed:

```

 $r_1$  : [ $>>$ person(John)]
 $r_2$  : [ $>>$ creditEligible(John)]
 $r_3$  : [ $r_3$ : person  $\Rightarrow$  contractuallyCapable,  $r_1$  :  $>>$ person(John)]
 $r_4$  : [ $>>$ mentalCondition(John)]
 $r_5$  : [ $r_5$ : mentalCondition  $\Rightarrow$   $\neg$  contractuallyCapable,  $r_4$  :  $>>$ mentalCondition(John)]

```

$r_6: [r_6: \text{mentalCondition} \Rightarrow \neg \text{creditEligible}, r_4 : >> \text{mentalCondition}(\text{John})]$
 $r_7: [r_7: \neg \text{creditEligible} \Rightarrow \neg \text{grantCredit}, r_6: \text{mentalCondition} \Rightarrow \neg \text{creditEligible}, r_4: >> \text{mentalCondition}(\text{John})]$
 $r_8: []$

After Line 7 we are only interested in rule which contradict each other. In case of T that is r_2 and r_6 as well as r_3 and r_5 . Since r_2, r_6, r_3, r_5 all have valid paths which activates the rules we can consider those two pairs with their corresponding paths as M_1 :

$M_1 :$

$r_2 : >> \text{creditEligible}(\text{John}),$
 $r_6 : \text{mentalCondition} \Rightarrow \neg \text{creditEligible},$
 $r_4 : >> \text{mentalCondition}(\text{John})$

$M_2 :$

$r_3 : \text{person} \Rightarrow \text{contractuallyCapable},$
 $r_1 : >> \text{person}(\text{John}),$
 $r_5 : \text{mentalCondition} \Rightarrow \neg \text{contractuallyCapable},$
 $r_4 : >> \text{mentalCondition}(\text{John})$

3.3.1.2 Finding quasi inconsistencies

The Algorithm 3 describes how MQI can be found in a FCL theory. It is structural similar to Algorithm 2.

Algorithm 3: Computation of minimal quasi inconsistent subsets in fcl

Input : Theory $T = (K, L)$
Output: MQI(T)

```

1  $mqis \leftarrow \emptyset$ 
2 for  $i$  from 1 to  $|K|$  do
3    $P_i \leftarrow \emptyset$ 
4 foreach  $\alpha \in K$  do
5    $P_\alpha = \{M_1, \dots, M_m \mid M \subseteq K\}$ 
6 foreach  $\alpha \in K$  do
7   if  $\exists x$  s.t.  $\beta_{h_x} == \neg\beta_{h_\alpha}$  then
8     foreach  $P' \in P_\alpha$  do
9       foreach  $P'' \in P_x$  do
10         $B_{P'} = \{\beta_1, \dots, \beta_i \mid \beta \in P'\}$ 
11         $B_{P''} = \{\beta_1, \dots, \beta_j \mid \beta \in P''\}$ 
12         $activationSet \leftarrow \{\{B_{P'}\} \cap \{B_{P''}\}\}$ 
13        if  $\{\{P'\} \cup \{activationSet\}\} \vdash \alpha \wedge$ 
14         $\{\{P''\} \cup \{activationSet\}\} \vdash x$  then
15           $mqis = mqis + \{\{P'\} \cup \{P''\}\}$ 

```

Like in finding MI for each contradicting pair of rules their paths are computed (Line 5). But in contrast to Algorithm 2 the paths are not checked if they activate their corresponding rule.

Each individual path of the rule is combined with every path from the contradicting rule. Now the intersection of each literal in each antecedent which has no further occurrence as the outcome of a rule is made. This intersection consists only of literals. For all of these literals are facts generated and stored into a set (Line 10-12). This set is considered as an activation set, when both rules are activated through the corresponding path and the activation set (Line 13 and 14).

When this requirement is met the two paths combined is an MQI and stored into the output variable (Line 15).

Example 12 (Finding an MQI in FCL). Consider following FCL theory:

$$\begin{aligned} r1 &: a, f \rightarrow c \\ r2 &: h, d \rightarrow \neg c \\ r3 &: b \rightarrow d \\ r4 &: b \rightarrow f \\ r5 &: a \Rightarrow h \end{aligned}$$

Since we are only interested in checking if contradicting rules are a MQI in the theory only the paths for $r1$ and $r2$ are checked resulting in the following paths.

$$\begin{aligned} r1 &: [r1 : a, f \rightarrow c; r4 : b \rightarrow f] \\ r2 &: [r2 : h, d \rightarrow \neg c; r3 : b \rightarrow d; r5 : a \Rightarrow h] \end{aligned}$$

Now each antecedent from both paths which has no further occurrence as an outcome occurrence is taken and the intersection of those two sets is build:

For $r1$: $\{a, b\}$

For $r2$: $\{a, b\}$

Resulting in the following intersection: $L = \{a, b\} \cap \{a, b\} = \{a, b\}$

For each literal in L a fact is constructed ($r6$ and $r7$) and a new theory T_q is build with the paths from the contradicting rules:

$$\begin{aligned} r1 &: a, f \rightarrow c \\ r2 &: h, d \rightarrow \neg c \\ r3 &: b \rightarrow d \\ r4 &: b \rightarrow f \\ r5 &: a \Rightarrow h \\ r6 &: >> a \\ r7 &: >> b \end{aligned}$$

This theory T_q is now checked with the normal MI solver in 3.3.1.1. If the results contains an inconsistency the combined paths is an quasi inconsistency due to the reason that there is one activation set which activates two contradicting rules. Otherwise the two paths are not quasi inconsistent because the activation

set build by the algorithm is not activating the two negating rules and therefore not causing an inconsistency.

3.3.1.3 Limitations

The solver has a limitation which it has inherited from the SPINdle library. It is not possible to assign variables to literals that means the solver can not distinguish between $a(X)$ and $a(Y)$.

Another restriction is that the computation of MQI does not consider literal variables. So if they occur in the theory MQI involving literal variable or literal boolean functions will not be found. An exception to this limitation is that SPINdle first sets the type of these function as normal literal. So the solver will probably find MI or MQI with literal boolean functions but only because they are considered as normal literals and not because of their meaning. An example for this can be seen below.

Example 13 (MQI with Literal Boolean Functions).

$$\begin{aligned} r_1 : \$@val>0\$ &\rightarrow x, \\ r_2 : \$@val>0\$ &\rightarrow \neg x, \end{aligned}$$

Example 13 shows a MQI where a literal function is involved. This kind of MQI will be found by the solver because the literal boolean function $\$@val>0\$$ is seen as a normal literal and due to the reason that this „literal“ occurs in r_1 and r_2 which have an contradicting output.

3.3.2 Declarative Process Model

In order to develop an algorithm to find MI or MQI in DECLARE a object model with a parser for the language has to be implemented.

In Figure 2 the class diagram for the DECLARE language is shown.

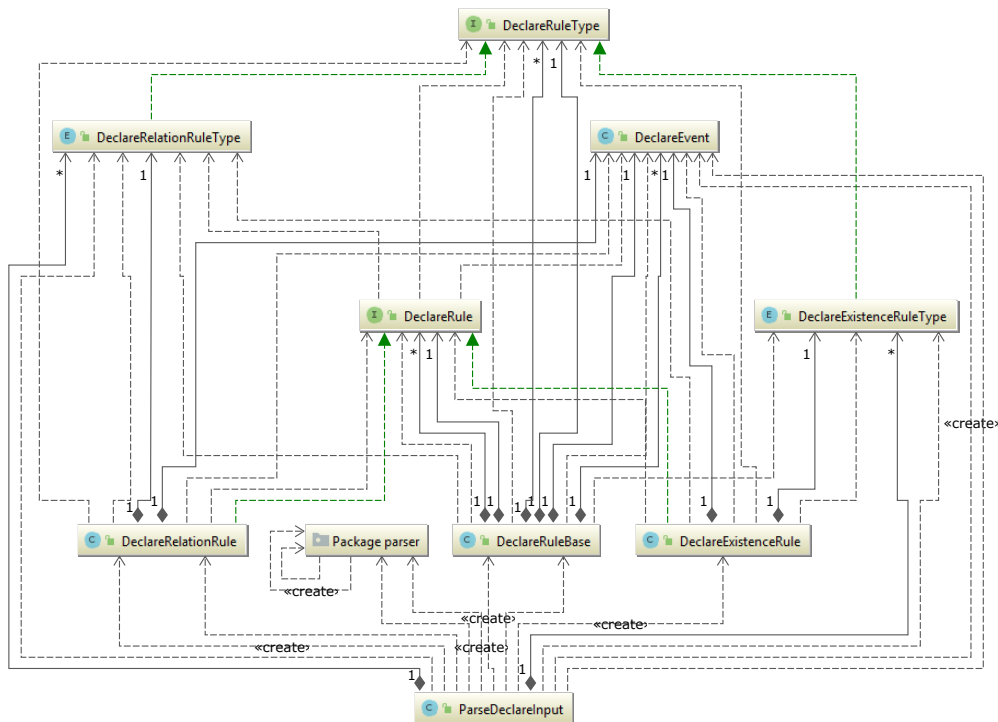


Figure 2 Declare language class diagram

Basically the class *ParserDeclareInput* uses the package *Parser* to parse a *Declare* input file and transfers it into a *DeclareRuleBase* object. This object contains *DeclareRule* objects which either can be *DeclareRelationRule* or *DeclareExistenceRule* objects.

The *DeclareExistenceRule* class implements the DECLARE Existence templates described in 2, whereas the *DeclareRelationRule* class the implementation of the DECLARE Relation templates is, which has been defined in Table 3 and 4.

The *Parser* package was implemented by using ANTLR (3.1). The Parser is able to parse the DECLARE language described in 2.1.2 with the restriction that every event has to be enclosed with single or double quotes.

3.3.2.1 Transferring to Graph

As a first step to computing the inconsistencies within a DECLARE constraint set, the set must be transformed into a graph on which the computation will be made. In the following an example set is given.

Example 14.

$$\mathcal{C}_1 = \{ \text{CHAINRESPONSE}(a, b), \quad \text{RESPONSE}(b, d), \\ \text{CHAINSUCCESSION}(b, c), \quad \text{ALTERNATERESPONSE}(d, e), \\ \text{CHAINRESPONSE}(e, c), \quad \text{PRECEDENCE}(d, c), \\ \text{NOTSUCCESSION}(a, c) \quad \}$$

In order to transform this every rule must be processed and stored within a graph. For each relational rule the first and second event is stored as a node and the type of the template is added as an edge between those two nodes with the type as the edge weight. The edge weight is stored as a set of relation template type because there can be multiple constraints with the same events but different types.

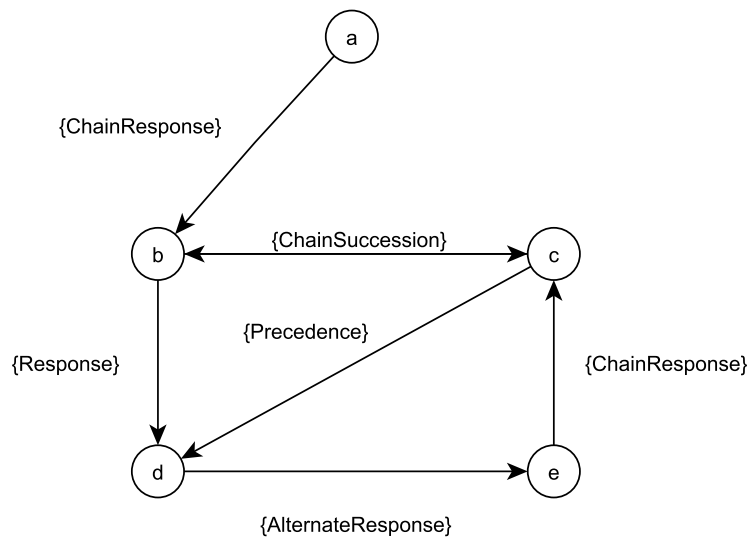


Figure 3 Transformed \mathcal{C}_1

The negation relation rules are not stored within the graph, because those are the main constraints that can cause an inconsistency within the set, additionally these constraints state that between the nodes can not be an edge connecting these.

For example the constraint CHAINRESPONSE(a,b) is stored in the graph (Figure 3) as node a and b with an edge that has CHAINRESPONSE as its weight. Whereas the rule NOTSUCCESSION(a,c) is not stored because it is a negation relation template and is therefore not added into the graph.

It has to be noted, that only forward templates can be stored in this way. For all PRECEDENCE templates the directed edge will start at the second node and for all SUCCESSION templates two directed edges are created due to the fact that both events are a needed factor to satisfy the constraint.

Generally the creation of edges is based upon the circles at the events in the graphical explanation of the relation templates in Table 3.

3.3.2.2 Algorithm MQIS in Declare

In order to compute all inconsistencies in the constraint set \mathcal{C}_1 , the program loops over every negation template (Algorithm 4, Line 3).

For every negation template the paths for event A are computed.

Algorithm 4: Computation of minimal quasi inconsistent subsets [CDD19a]

Input : Set of constraints \mathbf{C}
Output: MQI(\mathbf{C})

```

1  $mqis \leftarrow \emptyset$ ;
2  $compConstraints = findComplements()$ ;
3 foreach  $n:compConstraints$  do
4    $\alpha \leftarrow n.activation$ ;
5    $\omega \leftarrow n.reactionTask$ ;
6    $\mathbf{P} = findPaths(\alpha, \bar{\omega}) \cup findPaths(\bar{\omega}, \alpha)$ ;
7   foreach  $P:\mathbf{P}$  do
8     if  $\alpha \cup n \cup P^C \models \perp$  then
9        $mqis \leftarrow mqis \cup n \cup P^C$ ;

```

In \mathcal{C}_1 is only one negation template, namely NOTSUCCESSION(a, b). In this case

all paths from event a are computed using the Finding Paths algorithm. Resulting in following paths:

1. $\{c = \{[\text{CHAINRESPONSE}(a, b) \rightarrow \text{CHAINSUCCESION}(b, c)]\}\}$
2. $\{c = \{[\text{CHAINRESPONSE}(a, b) \rightarrow \text{RESPONSE}(b, d) \rightarrow \text{ALTERNATERESPONSE}(d, e) \rightarrow \text{CHAINRESPONSE}(e, c)]\}\}$
3. $\{d = \{[\text{CHAINRESPONSE}(a, b) \rightarrow \text{RESPONSE}(b, d)]\}\}$
4. $\{d = \{[\text{CHAINRESPONSE}(a, b) \rightarrow \text{CHAINSUCCESION}(b, c) \rightarrow \text{PRECEDENCE}(d, c)]\}\}$
5. $\{e = \{[\text{CHAINRESPONSE}(a, b) \rightarrow \text{RESPONSE}(b, d) \rightarrow \text{ALTERNATERESPONSE}(d, e)]\}\}$
6. $\{e = \{[\text{CHAINRESPONSE}(a, b) \rightarrow \text{CHAINSUCCESION}(b, c) \rightarrow \text{PRECEDENCE}(d, c) \rightarrow \text{ALTERNATERESPONSE}(d, e)]\}\}$
7. $\{b = \{[\text{CHAINRESPONSE}(a, b)]\}\}$

The first letter indicates which end event the path has, followed by the actual path. After the negation constraint it is only interesting if there is a path between a and c . Since Path 1 and 2 with c as the end event, they can be further checked if they violate requirements of the negation constraint. Here both paths violate the $\text{NOTSUCCESION}(a, b)$ constraint and therefore can be returned as a set of MQI:

$\text{MQI}_1 =$

$\{\text{CHAINRESPONSE}(a, b), \quad \text{CHAINSUCCESION}(b, c),$
 $\text{NOTSUCCESION}(a, b)\}$

$\text{MQI}_2 =$

$\{\text{CHAINRESPONSE}(a, b), \quad \text{RESPONSE}(b, d),$
 $\text{ALTERNATERESPONSE}(d, e), \quad \text{CHAINRESPONSE}(e, c),$
 $\text{NOTSUCCESION}(a, b)\}$

3.3.2.3 MIS in Declare

When searching for MI a different approach was made (shown in Algorithm 5): First there has to be an existence template with one event x . With this template it

can be assumed that this event has to be in the instance (or anything similar). Now all paths from this event x are computed and stored with the Finding Paths (Line 5). The next step is similar to 3.3.2.2, for all negation templates with events a and b is checked if there are paths from x to a and x to b (Line 9-13). When for both events paths exist it can be presumed that events a and b will occur in the instance and therefore violate the negation constraint and considered and MI (Line 14-17).

Algorithm 5: Computation of minimal inconsistent subsets

Input : Set of constraints \mathbf{C}

Output: $\mathbf{MI}(\mathbf{C})$

```

1   $mis \leftarrow \emptyset;$ 
2   $exConstraints = getExistenceConstraints();$ 
3   $negConstraints = getNegationConstraints();$ 
4  foreach  $e:exConstraints$  do
5     $P_e = findPaths(e);$ 
6    foreach  $n:negConstraints$  do
7       $firstPath \leftarrow \emptyset;$ 
8       $sndPath \leftarrow \emptyset;$ 
9      foreach  $p : P_e$  do
10     if  $p.endEvent().equals(n.getFirstEvent())$  then
11        $firstPath = firstPath + p$ 
12     if  $p.endEvent().equals(n.getSecondEvent())$  then
13        $sndPath = sndPath + p$ 
14     if  $!firstPath.isEmpty() \ \&\& \ !sndPath.isEmpty()$  then
15       foreach  $p_1 : firstPath$  do
16         foreach  $p_2 : sndPath$  do
17            $mis = mis + \{p_1 + p_2\}$ 

```

Example 15 (MI in DECLARE). The following declarative process model is an modification of the constraint set in 3.3.2.2. In addition \mathcal{C}_2 contains an existence

template which requires the event a to be present in an instance. The paths and figures regarding \mathcal{C}_1 are identical for \mathcal{C}_2 .

$$\mathcal{C}_2 = \{ \text{CHAINRESPONSE}(a, b), \quad \text{RESPONSE}(b, d), \\ \text{CHAINSUCCESSION}(b, c), \quad \text{ALTERNATERESPONSE}(d, e), \\ \text{CHAINRESPONSE}(e, c), \quad \text{PRECEDENCE}(d, c), \\ \text{NOTSUCCESSION}(a, c) \quad \text{PARTICIPATION}(a) \}$$

After Algorithm 5 the paths for the event a in the PARTICIPATION constraint are computed and stored (cf. 3.3.2.2).

Now for each negation constraint in \mathcal{C}_2 the reachability of the events contained in the negation constraint is checked. Since there is only one negation constraint (NOTSUCCESSION(a, c)) this is the only one to be checked. Since the event a is reachable through the PARTICIPATION(a) constraint and there are paths which have the event c as their end event, the algorithm will find the inconsistencies in \mathcal{C}_2 resulting in the following MIs:

- $\text{MI}_1 = \{ \text{PARTICIPATION}(a), \text{CHAINRESPONSE}(a, b), \\ \text{CHAINSUCCESSION}(b, c) \}$
- $\text{MI}_2 = \{ \text{PARTICIPATION}(a), \text{CHAINRESPONSE}(a, b), \text{RESPONSE}(b, d), \\ \text{ALTERNATERESPONSE}(d, e), \text{CHAINRESPONSE}(e, c) \}$

3.3.3 DMN

3.3.3.1 Finding Overlaps

In order to be able to determine if rows are overlapping we have to check for each column if there are overlapping rows. This means it has to be identified if there is a specific input for which two or more rows can be activated. In this thesis a similar approach to Calvanese et al. [CDL⁺16] was made. Calvanese made a similar approach to find overlapping rows in decision tables. But in contrast this approach focuses of finding overlapping rows not overlapping sets in decision tables which is more reasonable attempt to find MI due to the reason that only two rows can contradict each other either in an MI or MQI.

Boolean

Finding contradicting or overlapping boolean values is realized with the basic boolean operators:

<i>a</i>	<i>b</i>	$(a \wedge b) \vee \neg(a \vee b)$
<i>False</i>	<i>False</i>	<i>True</i>
<i>True</i>	<i>False</i>	<i>False</i>
<i>False</i>	<i>True</i>	<i>False</i>
<i>True</i>	<i>True</i>	<i>True</i>

Table 10 Truth Table

Basically the term evaluates if the two boolean values *a* and *b* are the same. In Java this term was implemented through the *equals* function which checks if two values are the same.

Furthermore booleans can cause inconsistency within decision tables, when they contradict each other. In other words when the terms evaluates for two output values to *False* values contradict each other and can cause an inconsistency.

Number Comparison

In case the type of the current input column is a numeric type, each value of a row for this column has to be compared to each other value in the column. The following table show each boolean functions to compare the possible numeric values in decision tables.

$A \setminus B$	$<$	$=$	$>$	$[A_1..A_2]$
$<$	$(A < B) \vee (B < A)$	$A < B$	$(A \neq B) \wedge ((A < B) \vee (B > A))$	$A_1 < B$
$=$	$B < A$	$A == B$	$B > A$	$(A_1 \leq B) \wedge (A_2 \geq B)$
$>$	$(B \neq A) \wedge ((B < A) \vee (A > B))$	$A > B$	$(A > B) \vee (B > A)$	$A_2 > B$
$[B_1..B_2]$	$B_1 < A$	$(B_1 \leq A) \wedge (B_2 \geq A)$	$B_2 > A$	$((A_1 \geq B_1) \wedge (A_1 \leq B_2)) \vee ((A_2 \geq B_1) \wedge (A_2 \leq B_2)) \vee ((B_1 \geq A_1) \wedge (B_1 \leq A_2)) \vee ((B_2 \geq A_1) \wedge (B_2 \leq A_2))$

Table 11 Comparison of Numbers in DMN Tables

But to compare each value with each other value in the column is inefficient due to the reason that the complexity rises factorial. The complexity in this case would be $O(x) = \prod_{i=1}^x i$ where x is the number of rows in the decision table. To decrease the complexity another approach was made. All values of a numeric type column were ordered and transferred to number line.

Example 16. Numeric values: $\{> 11, < 9, [-6..1]\}$

Corresponding number line:

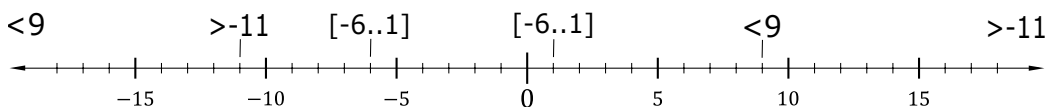


Figure 4 Number Line

Now to find all overlaps, there is an iterator through the number line which adds all corresponding matches to an set and stores the values. For example for

the value > -11 all remaining values on the number line are added to the set, or for the value < 9 all values iterated over before are added to the set.

The only time a value has to be compared the way described in Table 11 is, when to values are attached to the same number on the number line. Leaving the optimal complexity around $O(x) = x$ where x is the number of rows we have. The worst case would be the same complexity as before, but for it to happen all base values must be the same, which would be unlikely.

Dates

For finding the overlapping rows in a column with a date type a much simpler approach was made. Similar to Table 11 every value has to be compared to each other value in the column.

It has to be noted, that a similar approach to Example 16 can be made where each date variable is added to a line and to find each overlap the same approach as before is made.

Strings

Comparing strings is relatively simple, when they are equal there is an overlap otherwise not.

But one additional property of strings in DMN is that they can be variables which can also be negated. When a simple string is stored the characters must be surrounded by quotes, otherwise it is considered a variable.

The negation of variables is realized through the *not()* function. In order to check if there are output string variables which contradict each other it has to be checked if there is for each non negated variable a negated version. When there are this can lead to inconsistency within the decision table.

3.3.3.2 Algorithm for one Table

Algorithm 6 shows how the finding of MI for one decision table is implemented. As input the algorithm takes a decision table with I as all input columns and O as all output column.

The first step is to find all contradicting output values for the row r for each

output column O_i in O .

Algorithm 6: Computation of minimal quasi inconsistent subsets in one decision table

Input : DMN Decision Table $T = (I, O)$

Output: MI(T)

```

1   $mi \leftarrow \emptyset$ 
2   $pairs \leftarrow \emptyset$ 
3  foreach  $O_i \in O$  do
4  |   foreach  $r \in O_i$  do
5  | |   foreach  $r' \in O_i | r \wedge r' \models \perp$  do
6  | | |    $pairs = pairs + \{r + r'\}$ 
7  foreach  $I_i \in I$  do
8  |    $tmpPairs \leftarrow \emptyset$ 
9  |   foreach  $r \in I_i$  do
10 | |   foreach  $r' \in I_i | r.isOverlapping(r') == true \wedge \{r + r'\} \in pairs$  do
11 | | |    $tmpPairs = tmpPairs + \{r + r'\}$ 
12 |    $pairs = tmpPairs$ 
13  $mi \leftarrow pairs$ 

```

Followed by going through each input column and their corresponding row values to check if there are input values that will activate a pair of rows. This realized through *isOverlapping* function which is implemented for each type of value occurring in DMN. The details on how it works for each type is outlined in the Finding Overlaps section.

Additionally it is checked if the overlaps occur in the *pairs* set to filter out all negligible pairs of rows.

After each iteration over an input column the *pairs* set is replaced with the *tmpPairs* set. This is because we want do eradicate all not overlapping rows because those are not able to be contradicting each other caused by the reason that there can not be a input value set which will activate those rows. This replacement of the *pairs* value will most likely to decrease the row pairs to check.

When all input columns have been checked the *pairs* variable will contain all MI in the decision table. The result is then returned in the *mi* variable.

Example 17 (Inconsistent Decision Table). Consider Table 6 as our input table for the algorithm.

First each output column is checked for contradicting outputs. In this case three row pairs are stored into the variable:

$$\begin{aligned} pairs = \{ & p_1 = [Row1, Row2], \\ & p_2 = [Row1, Row3], \\ & p_3 = [Row1, Row4] \} \end{aligned}$$

Now for each of the pairs it is checked if the corresponding input values are overlapping. The pair p_1 is deleted right after the first input column „Credit Rating“ because the values contradict each other so that the rows could never be activated at the same time. The other two pairs are not deleted as their input values for this columns are empty.

In the next column „Credit Card Balance“ both row pairs still overlap p_2 with the input value 10000 and p_3 for the reason that the input value for Row 4 is empty.

Last the „Loan Balance“ is checked and still both pairs overlap p_3 with the input value range of [40000..49999] and p_2 for the reason that the input value for Row 3 is empty.

Now that all input columns have been iterated, the pairs remaining in the variable can be regarded as quasi inconsistent, since there is at least one input set for each pair that activates the rows contained therein.

3.3.3.3 Limitations

Currently the solver for decision tables is not capable of finding quasi inconsistencies in multiple depending decision tables. This is a complex problem which requires future work.

One of the problems with depending tables is shown in Example 18.

Example 18 (Depending Decision Tables). Consider Table 12 the first table to be evaluated. Since the input value overlaps for each row, each row can be activated at the same time resulting in the fact that the Input1 and Input2 variable have multiple values. (Input1 = b,c and Input2 = d,e).

	Input	Output	
C	Input	Input1	Input2
1	a	b	-
2	a	c	-
3	a	-	d
4	a	-	e

Table 12 First table

As Table 13 input columns are depending on the output variables of Table 12 and the output variables have multiple values the two rows are activated too. This results in a contradiction in this table (f and not(f)).

	Input		Output
C	Input1	Input2	Output
1	b	d	f
2	c	e	not(f)

Table 13 Dependent Quasi Inconsistent Table

Which would then be a quasi inconsistency:

$$\begin{aligned}
 \text{MQI} = \{ & \\
 & A = \{ \text{Input} = a \}, \\
 & C = \{ \text{Row 1 @ Table 13}, \\
 & \quad \text{Row 2 @ Table 13}, \\
 & \quad \text{Row 1 @ Table 12}, \\
 & \quad \text{Row 2 @ Table 12}, \\
 & \quad \text{Row 3 @ Table 12}, \\
 & \quad \text{Row 4 @ Table 12} \} \}
 \end{aligned}$$

The problem with finding with quasi inconsistencies in multiple decision tables like in Example 18 is that the rows where the contradicting values occur is dependent on various other tables. This case would require the solver to replace the dependent input columns with the lines of the original table that have the desired output values. But this would require additional merging steps to ensure that the rows stays semantically the same. This would result in swapping out the values for the corresponding rows:

	Input		Output
C	Input1	Input2	Output
1	Row 1 @ Table 12	Row 3 @ Table 12	f
2	Row 2 @ Table 12	Row 4 @ Table 12	not(f)

Table 14 Dependent Quasi Inconsistent Table

The problem is revealed here where both rows refers to the same table where they have the same input columns. In case of the example this would be no problem but when the input values for the two rows that where inserted would not be the same this would be an issue due to the reason that DMN does not allow you to have multiple values for one input value.

Another way would be to evaluate each table first which is not dependent and step-wise evaluate the other tables in order to assess the dependent tables correctly. So the evaluation would compute sets of rows which can occur together as well as the set of their corresponding output values. This is much like the conduct of Example 18.

3.4 BPMN

The main objective for BPMN in this thesis is to check models against DECLARE constraints. A similar concept was also developed by Mishra et al. for Semantic of Business Vocabulary and Business Rules (SBVR)[MS15] .

In order to do that is has to be specified how each template is checked against a BPMN model. First the Existence templates has to be defined

- $\text{PARTICIPATION}(A)$: A vertex with the label A occurs in the model.
- $\text{ATMOSTONE}(A)$: When a vertex with the label A occurs in the model there is no path from A to A
- $\text{INIT}(A)$: A start event with the label A occurs in the model.
- $\text{END}(A)$: An end event with the label A occurs in the model.

Additionally the Relation templates are defined:

- $\text{RESPONDEDEXISTENCE}(A, B)$: When a vertex with the label A occurs in the model there must be a path from A to B
- $\text{COEXISTENCE}(A, B)$: When a vertex with the label A occurs in the model there must be a path from A to B , when a vertex with the label B occurs in the model there must be a preceding path from B to A
- $\text{RESPONSE}(A, B)$: When a vertex with the label A occurs in the model there must be a sequence path from A to B
- $\text{PRECEDENCE}(A, B)$: When a vertex with the label B occurs in the model there must be a preceding sequence path from B to A
- $\text{SUCCESSION}(A, B)$: When a vertex with the label A or B occurs in the model there must be a sequence path from A to B additionally there must be a preceding sequence path from B to A .
- $\text{ALTERNATERESPONSE}(A, B)$: When a vertex with the label A occurs in the model there must be a sequence path from A to B and their must not be a sequence path from A to A

- $\text{ALTERNATEPRECEDENCE}(A, B)$: When a vertex with the label B occurs in the model there must be a preceding sequence path from B to A and their must not be a sequence path from B to B
- $\text{ALTERNATESUCCESSION}(A, B)$: When a vertex with the label A or B occurs in the model there must be a sequence path from A to B and there must be a preceding sequence path from B to A . Additionally there must not be a sequence path from A to A and B to B
- $\text{CHAINRESPONSE}(A, B)$: When a vertex with the label A occurs in the model one of the next vertices must be B
- $\text{CHAINPRECEDENCE}(A, B)$: When a vertex with the label B occurs in the model one of the preceding vertices must be A
- $\text{CHAINSUCCESSION}(A, B)$: When a vertex with the label A occurs in the model one of the next vertices must be B and when a vertex with the label B occurs in the model one of the preceding vertices must be A .

Not all SUCCESSION and COEXISTENCE constraints restrictions are checked exactly like described because when one restriction is met the other is inherited. For example with the template $\text{SUCCESSION}(A, B)$ if there is an vertex in the model with the label A and there is a sequence path from A to B there must be also a preceding path from B to A . This is inherited from the existence of a path from A to B .

Lastly there are negation relation templates:

- $\text{NOTCOEXISTENCE}(A, B)$: When a vertex with the label A occurs in the model there must not be a path from A to B and when a vertex with the label B occurs in the model there must not be a preceding path from B to A
- $\text{NOTSUCCESSION}(A, B)$: When a vertex with the label A or B occurs in the model there must not be a sequence path from A to B additionally there must not be a preceding sequence path from B to A .
- $\text{NOTCHAINSUCCESSION}(A, B)$: When a vertex with the label A occurs in the model all of the next vertices must not B and when a vertex with the label B occurs in the model all of the preceding vertices must not be A .

3.4.1 Graph

Like the solver for DECLARE a graph based approach was made for BPMN.

Definition 16 (Graph for BPMN). Given a Business Process Model $M = (V, F)$, where V is the set of elements and F is the set of flows within the model. Flows are defined as $f = (N, t)$ where $N = v_1 \times v_2$ and t the type of flow (e.g. sequence or message flow).

Since this is basically aligned with the definition of a graph it can easily transformed into one: Graph $G=(A,E)$ where $A = \{v|v \in V\}$ and $E \subseteq A \times A$. In addition $\forall e \in E$ exists a type t which is taken from the the original flow f .

A example for the transformation of a BPMN model to a graph is shown in Example 19.

In the Definition 16 is defined, that for each element in a BPMN a node is created and that these nodes are connected through edges which have a type t . This type is corresponding to the flow type in the business process model (sequence flow or message flow).

3.4.2 Checking DECLARE Constraints in BPMN

When there is a BPMN model to check against DECLARE constraints, first the BPMN Model M is transformed into a graph (line 2). The function *generateGraph* creates the graph according to the definition Graph for BPMN.

After the graph is created each constraint t in the set \mathcal{C} is checked if the requirements are met within the model M with the help of the graph generated from M (line 3-12).

The check of requirements of the constraints are according to the specification listed in 3.4.

Additionally further information about the inconsistency is gathered depending on the type of the inconsistency. Following types are considered in the implementation:

Algorithm 7: Computation of minimal quasi inconsistent subsets in fcl

Input : Theory $I = (M, \mathcal{C})$
Output: $MI(I)$

```

1  $mi \leftarrow \emptyset$ 
2  $G = generateGraph(M)$ 
3 foreach  $t \in \mathcal{C}$  do
4   if  $t$  is existence template then
5     if  $!checkExistenceRequirements(t, G)$  then
6        $mi = mi + t$ 
7   else if  $t$  is relation template then
8     if  $!checkRelationRequirements(t, G)$  then
9        $mi = mi + t$ 
10  else if  $t$  is negation relation template then
11    if  $!checkNegationRelationRequirements(t, G)$  then
12       $mi = mi + t$ 

```

Inconsistency Type	Description
MissingA	Event A is missing
MissingB	Event B is missing
LoopA	Loop at event A
LoopB	Loop at event B
Precedence	Event B has no Event A as predecessor
Response	Event A has no Event B as Response
NoChainPredecessor	Event A is not a direct predecessor of Event B
ChainPredecessor	Event A is a direct predecessor of Event B
ChainSuccessor	Event B is a direct successor of Event A
PathsAB	There are paths between A and B
NoChain	There is no direct connection between Event A and B

Table 15 Description of Inconsistency Types

Not each inconsistency type occurs for each DECLARE template. This is because of the definition of the templates and the types. For example for the constraint $\text{PRECEDENCE}(A, B)$ the inconsistency type MissingB can never occur since for the constraint to be activated there has to be an vertex with the label B .

So in Table 16 for each constraint DECLARE template the possible inconsistency types are listed. In addition to the inconsistency types a path is added to the MI. Which kind of path for each inconsistency type is listed in Table 17.

It has to be noted that all inconsistency types listed in Table 15 are mostly for relation templates.

Inconsistency Type	Inconsistency Path
MissingA	All Paths from B
MissingB	All Paths from A
LoopA	The loop path at A
LoopB	The loop path at B
Precedence	All preceding paths from B
Response	All Paths from A
NoChainPredecessor	All direct predecessors from B
ChainPredecessor	The direct predecessor Event A
ChainSuccessor	The direct successor Event B
PathsAB	The paths between A and B
NoChain	The direct successor B

Table 17 Description of Paths with their Inconsistency Types

There are only two inconsistency types which can also occur for existence templates: MissingA and LoopA. Where MissingA can occur for PARTICIPATION, INIT, END, LoopA can only occur for ATMOSTONE. That comes from the reason that when there is a loop at A the event A can occur more than once and therefore violating the ATMOSTONE constraint.

Furthermore two more inconsistency types were added in order to point out the

DECLARE templates	Inconsistency types										
	MissingA	MissingB	LoopA	LoopB	Precedence	Response	NoChain	NoChainPredecessor	ChainPredecessor	ChainSuccessor	PathsAB
RESPONDEDEXISTENCE		✓									
COEXISTENCE	✓	✓									
RESPONSE						✓					
PRECEDENCE					✓						
SUCCESSION					✓	✓					
ALTERNATERESPONSE			✓			✓					
ALTERNATEPRECEDENCE				✓	✓						
ALTERNATESUCCESSION			✓	✓	✓	✓					
CHAINRESPONSE							✓				
CHAINPRECEDENCE								✓			
CHAINSUCCESSION							✓	✓			
NOTCOEXISTENCE											✓
NOTSUCCESSION											✓
NOTCHAINSUCCESSION									✓	✓	

Table 16 Inconsistency types for DECLARE relation templates

requirement of the type for the INIT and END templates. These are named NotInit and NotEnd aligning with their corresponding template. These types can only occur when the event is in the process model but the type does not meet the constraint (for INIT start event and for END end event).

Example 19 (BPMN checked with DECLARE).

Consider the following business process model in BPMN:

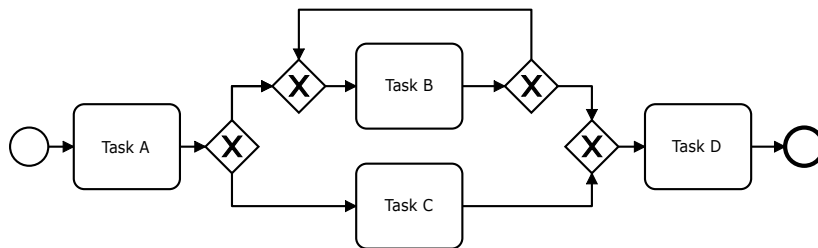


Figure 5 Example BPMN model

In combination with the following DECLARE constraint set.

$$\mathcal{C} = \{ \text{ALTERNATEPRECEDENCE}(\text{Task A}, \text{Task B}), \\ \text{NOTCOEXISTENCE}(\text{Task C}, \text{Task B}), \\ \text{NOTCHAINSUCCESSION}(\text{Task C}, \text{Task D}), \\ \text{ALTERNATERESPONSE}(\text{Task A}, \text{Task D}), \\ \text{CHAINRESPONSE}(\text{Task A}, \text{Task C}) \}$$

From the model in Figure 5 a graph is generated after the Definition 16. The corresponding graph is shown in Figure 6. This graph looks much like the model, but the gateways have been renamed and the edges have the flow type has a weight (the edge weight in the figure is omitted due to the reason that only sequence flows occur in the model).

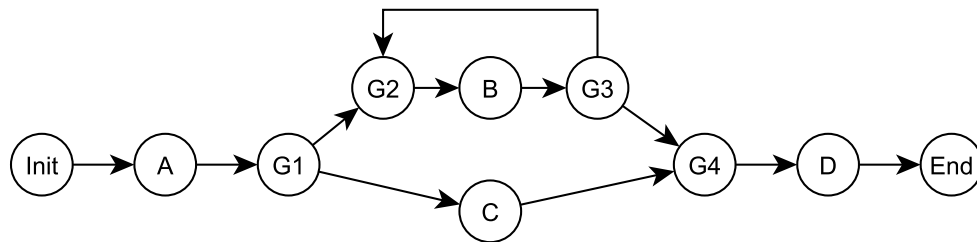


Figure 6 Graph for example model

When the graph is created, each constraint is checked to see if it is met in the graph. In order to see if the requirements of the constraint is met the Finding Paths algorithm is needed. For each constraint the activating entity (or entities) is taken and all paths from the event are computed. Each path is now checked if it aligns with the constraint or violates it. For example the for the constraint `CHAINRESPONSE(Task A, Task B)` it takes the Task A and computes the following paths:

- $p_1 = [\text{Task A, G1, G2, Task B, G3, G2, Task B}]^{12}$
- $p_2 = [\text{Task A, G1, G2, Task B, G3, G4, Task D, End}]$
- $p_3 = [\text{Task A, G1, Task C, G4, Task D, End}]$

Now the paths are checked and since p_3 satisfies the constraint it is not violated within the process model.

This is done for each constraint resulting in the model violating some of the constraints:

- $Ml_1 =$
`NOTCHAINSUCCESION(Task C, Task D),`
 Reason: ChainSuccessor,
 Path=[Task A, G1, Task C]
- $Ml_2 =$
`ALTERNATEPRECEDENCE(Task A, Task B),`
 Reason: LoopB,
 Path=[Task A, G1, G2, Task B, G3, G2, Task B]

¹²The algorithm stops when a task node occurs the second time

The Ml_1 consists of the NOTCHAINSUCCESSION(Task C, Task D) and the path violating it. The path is a violation of the constraint although there is a gateway G1 between Task C and Task D. Because the gateway will never occur in a sequence of tasks, so task D is the direct successor of task C in a sequence.

The ALTERNATEPRECEDENCE(Task A, Task B) constraint in Ml_2 on the other hand is violated because of loop at Task B which contradicts the requirement of the ALTERNATEPRECEDENCE template that the entity B can not occur twice before entity A occurs.

3.4.3 Limitations

Currently the nodes in the graph generated from the BPMN are based on the labels of the corresponding tasks. This can lead to a drawback when there are multiple tasks with the same label. This can cause problems when checking DECLARE constraints with labels that are occurring multiple times in the BPMN.

Also a limitation of the current implementation is that it is not possible to check constraints which involve normal tasks in combination with tasks which are in an expanded sub process. In the graph where the constraints are checked on there is no connection between the „normal“ vertices and the vertices located in the sub process. This is because the start event of the sub process is not connected via a sequence or message flow which causes the transformation algorithm to make no connection and considering the sub process as its own entity.

Chapter 4

Evaluation

For evaluation the aspects of finding MI, MQI and checking DECLARE constraints against BPMN models is done with one system configuration (CPU: Intel Xeon E3-1230v5 @ 3.40 GHz¹, RAM: 32GB DDR4 ECC @ 2133MHz, OS: Windows 10 Version 1803).

4.1 Business Rules

For the evaluation of the business rule solvers the runtime for each is collected as well as the regarding MI and MQI count for the test data sets.

4.1.1 Defeasible Logic

For FCL a random theory generator was implemented. This generator creates a FCL theory from different parameters:

- *ruleCount*: Number of rules
- *literalCount*: Number of literals
- *maxBodyLiteral*: Number of maximal body literals in strict and defeasible rules
- *pStrict*: Percentage of strict rules

¹<https://ark.intel.com/de/products/88182/Intel-Xeon-Processor-E3-1230-v5-8M-Cache-3-40-GHz-/>

- *pDefeasible*: Percentage of defeasible rules
- *pFacts*: Percentage of facts
- *pSuperiority*: Percentage of superiorities for contradicting defeasible rules

For the evaluation following parameters were fixed:

- *maxBodyLiteral* = 2
- *pStrict* = 30%
- *pDefeasible* = 50%
- *pFact* = 20%
- *pSuperiority* = 5%

The only things that were changed throughout the runs are the number of rules and the number of literals.

In Figure 7 the compute time for the generated theories is shown. It stands out that the computation time mostly is negligible, but the fewer the number of literals and the greater the number of rules the greater the computation time.

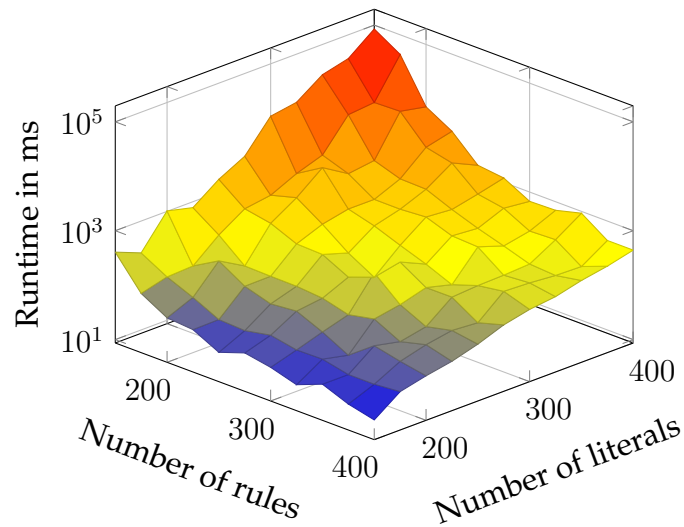


Figure 7 Runtime statistics for the FCL solver

This observation can also be made in Figure 8 and Figure 9. The fewer the number of literals and the greater the number of rules the greater the number of MI or MQI in the test data.

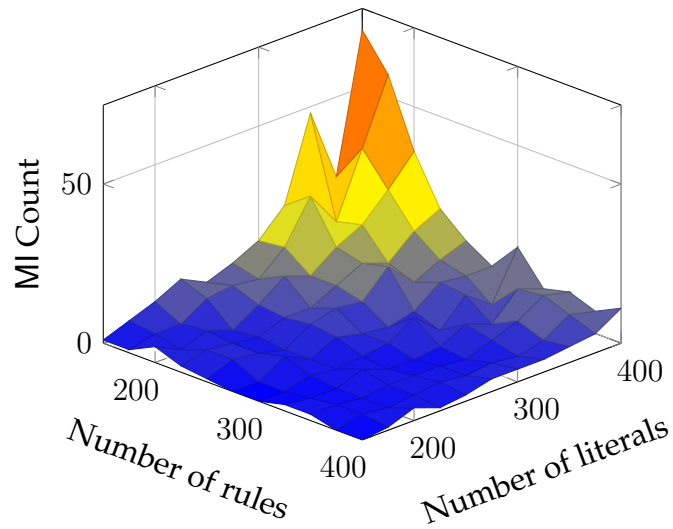


Figure 8 MI in FCL test data

This can be explained because the lower the amount of literals in a theory the higher the dependability of rules within the theory. This results from the fact that literals will occur more often in rules as well as their contradicting literal the lower the amount of available literals and the higher the amount of generated rules are.

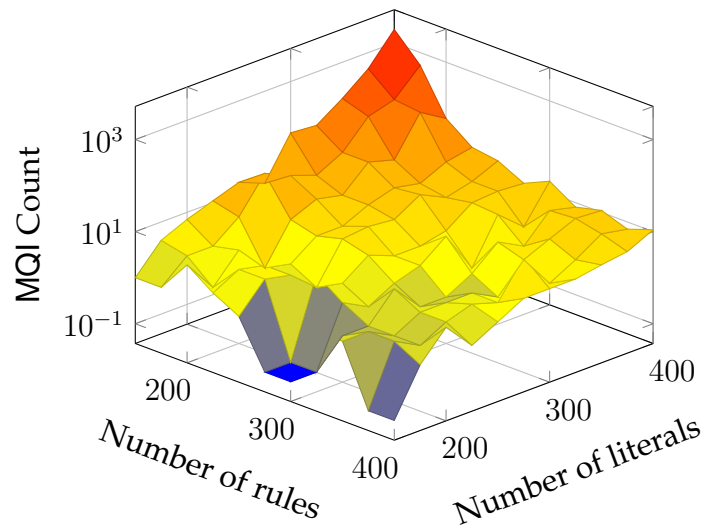


Figure 9 MQI in FCL test data

A connection can be made between the computational time and the amount of MI/MQI.

But this applies only to the FCL theory generator, which generates the rules in a randomly manner. Therefore the figures show only the results for this particular data set.

In addition to this configuration of the theory generator, three further test configuration runs were performed. The corresponding graphics are listed in the Appendix D. From these a few observations can be made:

- The higher the amount of facts, the lower the runtime (cf. Figure 16)
- The higher the amount of facts, the lower the amount of MQI (cf. Figure 18)
- The higher the max amount of body literals, the higher the runtime (cf. Figure 19)
- The higher the max amount of body literals, the lower the amount of MI (cf. Figure 20)
- The higher the max amount of body literals, the higher the amount of MQI (cf. Figure 21)
- Swapping the share of defeasible rules to the strict rules have no big impact on performance and amount of MI/MQI (cf. Appendix D.1)

These are observations that can be made, only if the data is randomly generated. In not generated theories the correlations can be similar to the shown one but do not have to.

4.1.2 Declarative Process Model

For the evaluation of the DECLARE solver the same dataset as in [CDD19a] was used:

- BPI challenge 2017². This data set contains an event log of a loan application process of a Dutch financial institute. The log is constituted of 1,202,267 events corresponding to 31,509 loan application cases.

²<https://www.win.tue.nl/bpi/doku.php?id=2017:challenge>

- BPI challenge 2018³. This data set contains an event log of a process at the level of German federal ministries of agriculture and local departments. The log comprises 2,514,266 events corresponding to 43,809 application cases.
- Sepsis 2016⁴. This data set contains an event log of a hospital process concerning sepsis, which is a life threatening condition. The log contains around 1000 cases with 15,000 events.

While it is the same dataset the outcome of found MQI is the same as in [CDD19a] as well as every other measure computed by the solver.

However the runtime is about 40% faster than in Corea et al.⁵

Log	BPI Challenge '17	BPI Challenge '18	Sepsis '16
Constraints	305	70	207
\mathcal{I}_{MI}^Q ⁶	28954	25303	7736
$\mathcal{I}_{MI^c}^Q$	2573.54	1763.43	848.70
\mathcal{I}_p^Q	87	62	53
\mathcal{I}_{mv}^Q	0.84	0.92	0.81
Runtime	15601ms	6959ms	2532ms

Table 18 Results of runtime experiments for the analyzed data-sets

This can be due to the reason of the hardware or software configuration difference between the test runs. The basic algorithm has not changed between the test runs which could have also impacted the performance.

Considering the runtime results it is possible with the algorithm developed in this thesis to find MQI in DECLARE data sets which are not only simple contradicting constrains as in [DCMMM17] by extending the scope to constraint chains which also can lead to inconsistencies.

³<https://www.win.tue.nl/bpi/doku.php?id=2018:challenge>

⁴<https://data.4tu.nl/repository/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460>

⁵BPI17: 43%, BPI18: 36%, Sepsis16: 42,18%

4.1.3 DMN

The evaluation for DMN was made with the dmn-generator implemented from Ramberger[Ram18].

The generator produces one kind of a decision table which consists of a variable number of rows and columns as well as one output column.

The input columns all are integer range inputs and the output type is strings acting as boolean variables.

Figure 10 shows the number of MQI depending on the number of input columns and the number of rows. Unlike the MQI count for the FCL generator the DMN generator produces decision tables with linear increasing MQI count for increasing number of rows. The different number of columns seems to have towards zero effect on the number of MQI in a decision table.

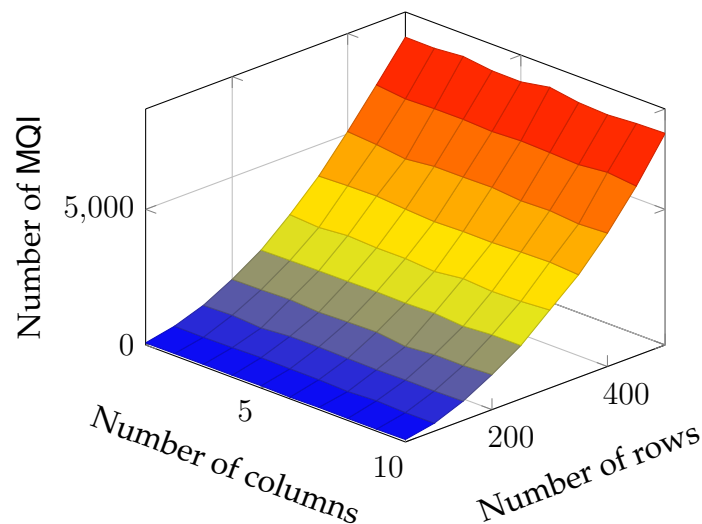


Figure 10 Number of MQI in DMN evaluation data set

Taking a deeper look in the algorithm of the dmn-generator it was evident that the integer variable creation is the cause of this behaviour. The integer ranges generated have an lower bound is between 0 and 15 and the upper bound is a number between 15 and 45. This results in that nearly all the generated rows overlap which causes the high amount of MQI shown in Figure 10. The amount of MQI would be higher if the output column would had more contradicting boolean variables.

This circumstance produces a sub optimal test data set which hurts the performance of the algorithm significantly. This leads to the runtime in Figure 11.

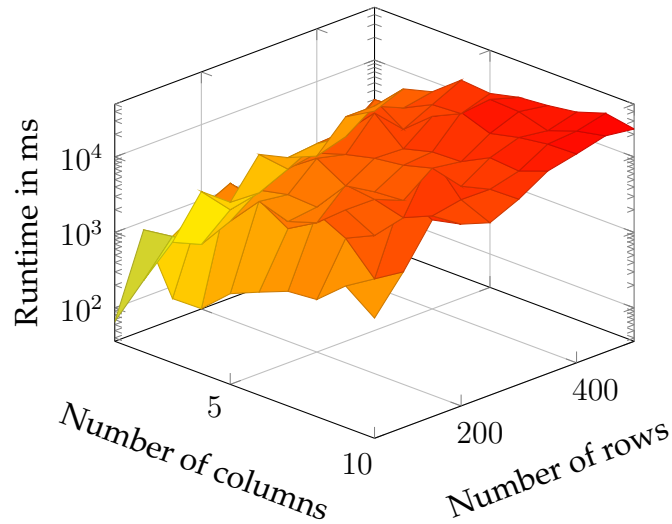


Figure 11 Runtime statistics for the DMN solver

With either the lowest amount of columns (1) or the lowest amount of rows (50) the runtime is relatively low with a maximum of 2.7s for 9 columns with 50 rows. With the increasing amount of rows and columns the runtime rises exponentially to a maximum of 26.5s for 10 columns with 450 rows.

Nonetheless the results of the test data set can be regarded positive due to the fact that the test data is highly sub optimal as explained before. It can be presumed that real data would behave different because normally decision trees do not have that high amount of overlapping rows which can lead to inconsistencies.

4.2 BPMN with DECLARE

For evaluating applying DECLARE constraints to business process models in BPMN an own approach was made, due to the lack of specific DECLARE constraints in combination with the corresponding business process model.

The business process model in Appendix C was used in combination with 28 DECLARE constraints which have been fabricated to test the solver. Each template type has occurred twice. One time were it is satisfied and one time were it is violated in the process model.

Part	Time (ms)
Initialization Time	62 ms
Solver Time	5576 ms

Table 19 Runtime for BPMN with DECLARE implementation

In Table 19 the runtime for the specific dataset in combination with the BPMN in Appendix C is shown. The initialization time includes the time needed to parse the BPMN and DECLARE file and the creation of a graph from the BPMN. The solver time stands for the time needed to check each of the constraints in the DECLARE file if they are satisfied in the BPMN. In Table 20 the amount of paths for the corresponding non satisfied constraints is shown. Here it is notable that the non satisfied `RESPONDEDEXISTENCE` and `COEXISTENCE` constraints have the highest amount of paths paired with their inconsistency type. This is due to the reason that for those two relation templates the corresponding inconsistency types will normally have the highest number of possible paths conflicting with the templates. This is because, unlike the other templates, these templates will also consider the message flows in BPMNs. Which results in the higher computation time needed by those templates. Whereas the `NOTCOEXISTENCE` template also considers message flows, but when this type of template is not satisfied there are most likely fewer amount of paths as the only paths conflicting will be between the events in the constraint.

DECLARE templates	Path count
RESPONDEDEXISTENCE	229
COEXISTENCE	1217
RESPONSE	3
PRECEDENCE	2
SUCCESSION	6
ALTERNATERESPONSE	6
ALTERNATEPRECEDENCE	2
ALTERNATESUCCESSION	8
CHAINRESPONSE	1
CHAINPRECEDENCE	1
CHAINSUCCESSION	1
NOTCOEXISTENCE	6
NOTSUCCESSION	6
NOTCHAINSUCCESSION	1

Table 20 Paths for each inconsistency

Chapter 5

Conclusion

This thesis documents the implementation of a library for finding inconsistencies and quasi inconsistencies in different business rule languages. Because these inconsistencies and quasi inconsistencies make business rule bases not usable this is important to detect those in knowledge bases. To solve the problem the prototype library contains an approach to find those inconsistencies and their origins. The library offers the possibility to check the rule bases at design time to find the contradictory elements within and their corresponding overlap conditions, as well as a quantitative analysis of the found inconsistencies through inconsistency measures. The individual error types in business rules defined by Smit et al. [SZB17] and the ability of the library for finding those errors is listed in Appendix A.

Yet there are still some limitations to the library regarding the ability to consider literal boolean functions properly as well as SPINdle restrictions such as the literal variables for the FCL solver. But the main impairment for the library is its inability to find quasi inconsistencies in multiple dependent decision tables. There has been a theoretical approach as discussed in 3.3.3.3, but has not yet been implemented. For future work this is the main objective to focus on. Furthermore the performance of the library can be optimized through parallel execution where applicable as well as minor improvements such as the implementation of differentiation of tasks with the same label in BPMN (cf. 3.4.3).

To the best of our knowledge, this is the first library to compute quasi inconsistencies in different formalisms, other researches have proposed different approaches to find basic inconsistencies such as Di Ciccio et al.[DCMMM17] for DECLARE, but no quasi inconsistencies which can also lead to serious problems within busi-

ness rule bases which can result in unusable declarative process models.

This is also yet the first implementation of a library which provides through additional information, such as different rule base measures, insights of the quality of a given rule base or model based on formalisms which are used in praxis. In addition, elemental measurements and inconsistency types can help characterize the culpability of certain elements and thus provide a helpful hand in resolving the inconsistencies in which they are involved.

Another feature is the ability to check DECLARE constraints against business process models in BPMN which is, as far as we know, also a first implementation and application for declarative process models.

Bibliography

- [ALR15] AA, Han van d. ; LEOPOLD, Henrik ; REIJERS, Hajo A.: Detecting Inconsistencies Between Process Models and Textual Descriptions. In: MOTAHARI-NEZHAD, Hamid R. (Hrsg.) ; RECKER, Jan (Hrsg.) ; WEIDLICH, Matthias (Hrsg.): *Business Process Management*. Cham : Springer International Publishing, 2015. – ISBN 978-3-319-23063-4, S. 90–105
- [BBG⁺90] BELL, J ; BROOKS, D ; GOLDBLOOM, E ; SARRO, R ; WOOD, J: Re-engineering case study analysis of business rules and recommendations for treatment of rules in a relational database environment. In: *Bellevue Golden: US West Information Technologies Group* (1990)
- [BMS16] BURATTIN, Andrea ; MAGGI, Fabrizio M. ; SPERDUTI, Alessandro: Conformance checking based on multi-perspective declarative process models. In: *Expert Systems with Applications* 65 (2016)
- [CA16] CHITTIMALLI, Pavan K. ; ANAND, Kritika: Domain-independent Method of Detecting Inconsistencies in SBVR-based Business Rules. In: *Proceedings of the International Workshop on Formal Methods for Analysis of Business Systems*. New York, NY, USA : ACM, 2016 (ForMABS 2016). – ISBN 978-1-4503-4214-8, 9–16
- [CD18a] COREA, Carl ; DELFMANN, Patrick: Supporting Business Rule Management with Inconsistency Analysis. In: *Proceedings of the BPM 2018 Industry Track co-located with the 16th International Conference on Business Process Management (BPM 2018)*, Sydney, Australia, September 09-14, 2018., 2018

- [CD18b] COREA, Carl ; DELFMANN, Patrick: A Tool to Monitor Consistent Decision-Making in Business Process Execution. In: *Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018 co-located with 16th International Conference on Business Process Management (BPM 2018), Sydney, Australia, September 9-14, 2018.*, 2018, 76–80
- [CDCDGM18] CECCONI, Alessio ; DI CICCIO, Claudio ; DE GIACOMO, Giuseppe ; MENDLING, Jan: Interestingness of traces in declarative process mining: The Janus LTLpf approach. In: *BPM, 2018*, S. 121–138
- [CDD19a] COREA, Carl ; DEISEN, Matthias ; DELFMANN, Patrick: Quasi-Inconsistency in Declarative Process Models. In: *Submitted to 17th Int. Conference on Business Process Management, BPM 2019*, 2019
- [CDD19b] COREA, Carl ; DEISEN, Matthias ; DELFMANN, Patrick: Resolving Inconsistencies in Declarative Process Models based on Cul- pability Measurement. In: *15. Internationale Tagung Wirtschaftsin- formatik, WI 2019*, 2019
- [CDL⁺16] CALVANESE, Diego ; DUMAS, Marlon ; LAURSON, Ülari ; MAGGI, Fabrizio M. ; MONTALI, Marco ; TEINEMAA, Irene: Semantics and Analysis of DMN Decision Tables. In: LA ROSA, Marcello (Hrsg.) ; LOOS, Peter (Hrsg.) ; PASTOR, Oscar (Hrsg.): *Business Process Management*. Cham : Springer International Publishing, 2016. – ISBN 978–3–319–45348–4, S. 217–233
- [CLRS09] CORMEN, T.H. ; LEISERSON, C.E. ; RIVEST, R.L. ; STEIN, C.: *Introduction to Algorithms*. MIT Press, 2009 (Com- puter science). <https://books.google.de/books?id=i-bUBQAAQBAJ>. – ISBN 9780262033848
- [DAPS09] DER AALST, Wil M. ; PESIC, Maja ; SCHONENBERG, Helen: Declarative workflows: Balancing between flexibility and sup- port. In: *Computer Science-Research and Development* 23 (2009), Nr. 2, S. 99–113
- [DCMM13] DI CICCIO, Claudio ; MECELLA, Massimo ; MENDLING, Jan: The effect of noise on mined declarative constraints. In: *International*

- Symposium on Data-Driven Process Discovery and Analysis* Springer, 2013, S. 1–24
- [DCMMM17] DI CICCIO, Claudio ; MAGGI, Fabrizio M. ; MONTALI, Marco ; MENDLING, Jan: Resolving inconsistencies and redundancies in declarative process models. In: *Information Systems* 64 (2017)
- [dmn18] ; Object Management Group (Veranst.): *Decision Model and Notation*. <https://www.omg.org/spec/DMN>. Version: 2018
- [GH13] GREGOR, Shirley ; HEVNER, Alan R.: Positioning and presenting design science research for maximum impact. In: *MIS quarterly* 37 (2013), Nr. 2
- [GK07] GHOSE, Aditya ; KOLIADIS, George: Auditing Business Process Compliance. In: KRÄMER, Bernd J. (Hrsg.) ; LIN, Kwei-Jay (Hrsg.) ; NARASIMHAN, Priya (Hrsg.): *Service-Oriented Computing – IC-SOC 2007*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2007. – ISBN 978–3–540–74974–5, S. 169–180
- [GM18] GRANT, John ; MARTINEZ, Maria V.: *Measuring Inconsistency in Information*. College Publications, 2018
- [Gra07] GRAHAM, Ian: *Business rules management and service oriented architecture: a pattern language*. John wiley & sons, 2007
- [HK⁺08] HUNTER, Anthony ; KONIECZNY, Sébastien u. a.: Measuring Inconsistency through Minimal Inconsistent Sets. In: *KR* 8 (2008)
- [HMPR04] HEVNER, Alan R. ; MARCH, Salvatore T. ; PARK, Jinsoo ; RAM, Sudha: Design science in information systems research. In: *Management Information Systems Quarterly* 28 (2004), Nr. 1, S. 6
- [HMPR08] HEVNER, Alan R. ; MARCH, Salvatore T. ; PARK, Jinsoo ; RAM, Sudha: Design science in information systems research. In: *Management Information Systems Quarterly* 28 (2008), Nr. 1, S. 6
- [KEP00] In: KNOLMAYER, Gerhard ; ENDL, Rainer ; PFAHRER, Marcel: *Modeling Processes and Workflows by Business Rules*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2000. – ISBN 978–3–540–45594–3, 16–29

- [MMv11] MAGGI, F.M. ; MOOIJ, A.J. ; VAN DER AALST, W.M.P.: User-Guided Discovery of Declarative Process Models. In: *2011 IEEE Symposium on Computational Intelligence and Data Mining*. 2011
- [MS15] MISHRA, Akanksha ; SUREKA, Ashish: A Graph Processing Based Approach for Automatic Detection of Semantic Inconsistency Between BPMN Process Model and SBVR Rules. In: PRASATH, Rajendra (Hrsg.) ; VUPPALA, Anil K. (Hrsg.) ; KATHIRVALAVAKUMAR, T. (Hrsg.): *Mining Intelligence and Knowledge Exploration*. Cham : Springer International Publishing, 2015. – ISBN 978-3-319-26832-3, S. 115–129
- [NCD19] NAGEL, Sabine ; COREA, Carl ; DELFMANN, Patrick: Effects of Quantitative Measures on Understanding Inconsistencies in Business Rules. In: *The 52nd Hawaii International Conference on System Sciences, Hawaii, USA, January 08-11, 2019.*, 2019
- [NPRS10] NELSON, Matthew L. ; PETERSON, John ; RARIDEN, Robert L. ; SEN, Ravi: Transitioning to a business rule management service model: Case studies from the property and casualty insurance industry. In: *Information & management* 47 (2010), Nr. 1, S. 30–41
- [Nut03] NUTE, Donald: Defeasible Logic. In: BARTENSTEIN, Oskar (Hrsg.) ; GESKE, Ulrich (Hrsg.) ; HANNEBAUER, Markus (Hrsg.) ; YOSHIE, Osamu (Hrsg.): *Web Knowledge Management and Decision Support*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2003. – ISBN 978-3-540-36524-2, S. 151–169
- [OMG13] OMG: *Information technology — Object Management Group Business Process Model and Notation*. Version: 2013. <https://www.iso.org/standard/62652.html>
- [Pes08] PESIC, Maja: Constraint-based workflow management systems: shifting control to users. (2008)
- [PTRC07] PEFFERS, Ken ; TUUNANEN, Tuure ; ROTHENBERGER, Marcus A. ; CHATTERJEE, Samir: A design science research methodology for information systems research. In: *Journal of management information systems* 24 (2007), Nr. 3, S. 45–77

- [Ram18] RAMBERGER, Henning: *Converting Decision Model and Notation Decision Tables to Formal Contract Language*. 2018
- [Roh05] ROHDE, Frank: Little decisions add up. In: *Harvard Business Review* 83 (6) 83 (2005), S. 24—26
- [SZB17] SMIT, K. ; ZOET, M. ; BERKHOUT, M.: Verification capabilities for business rules management in the Dutch governmental context. In: *2017 International Conference on Research and Innovation in Information Systems (ICRIIS)*, 2017. – ISSN 2324–8157, S. 1–6
- [Thi16] THIMM, Matthias: On the Compliance of Rationality Postulates for Inconsistency Measures: A More or Less Complete Picture. In: *Künstliche Intelligenz* (2016), August
- [Thi18] THIMM, Matthias: On the Evaluation of Inconsistency Measures. In: GRANT, John (Hrsg.) ; MARTINEZ, Maria V. (Hrsg.): *Measuring Inconsistency in Information* Bd. 73. College Publications, February 2018
- [VK04] VAISHNAVI, V. ; KUECHLER, W.: Design Research in Information Systems. (2004), Januar
- [VPHB16] VENABLE, John ; PRIES-HEJE, Jan ; BASKERVILLE, Richard: FEDS: a Framework for Evaluation in Design Science Research. In: *European Journal of Information Systems* 25 (2016), Jan, Nr. 1, 77–89. <http://dx.doi.org/10.1057/ejis.2014.36>. – DOI 10.1057/ejis.2014.36. – ISSN 1476–9344

Appendix A

Verification Capabilities

Detection abilities						
Duplicated Rules	Equivalent Rules	Subsumed Rules	Unnecessary Facts	Contradicting Conclusions	Overlapping Conclusions	Missing Rules
				✓	✓	

Table 21 Verification capability of library [SZB17]

Appendix B

Usage of Library

To be able to use the library only a Java runtime environment has to be installed, no other prerequisites are needed.

The library along with the source code is available at <https://gitlab.uni-koblenz.de/mdeisen/inconsistency-lib> or [inconsistency.matrose.de](https://gitlab.uni-koblenz.de/mdeisen/inconsistency-matrose) The inconsistency library comes as a executable jar which can be executed via the `java -jar` command:

```
C:\Users\User\code\>java -jar inconLib.jar
```

Usage:

Defeasible Logic:

```
java -jar inconLib.jar fcl <Filename>
```

Declare:

```
java -jar inconLib.jar declare <Filename>
```

DMN:

```
java -jar inconLib.jar dmn <Filename>
```

BPMN-Declare:

```
java -jar inconLib.jar _  
bpmn-declare <BPMN-Filename> <Declare-Filename>
```

When the `inconLib` is used this way it returns all of the minimal inconsistent subsets as well as the minimal quasi inconsistent subset when applicable.

Furthermore the library can be integrated into other java projects to use extended features of the library, e.g. the measures computed by the library. The classes to do this are listed below:

-
- **Defeasible Logic:** MISCompute (<FCL Theory>, <boolean computeQMIS>)
 - getResult (): Returns the MI
 - getQMIS (): Returns the MQI
 - getMIVdMeasure (): Returns the MI measure
 - getMIVcMeasure (): Returns the MI measure
 - getCMsharpMeasure (): Returns the MI measure

 - **DECLARE:** DeclareSolver (<Declare rule base>)
 - getResult (): Returns the MI
 - getQMISResult (): Returns the MQI
 - measures (): Returns the MI measures
 - getQmisMeasures (): Returns the MQI measures

 - **DMN:** TestDMN (<DMN File>)
Methods:
 - getInconsistencies (): Returns the MI for each table with row numbers
 - getExecTime (): Returns the execution time for computing the MI

 - **BPMN-DECLARE:**
CheckBpmnDeclare (<BPMN Model>, <Declare rule base>)
Methods:
 - getIncons (): Returns the MI with IDs
 - getInconsNames (): Returns the MI with Labels

Appendix C

BPMN

This Business Process model is taken from the Process Model Matching Contest 2015¹ and describes the enrollment process at the University of Hohenheim.

¹<https://ai.wu.ac.at/emisa2015/contest.php>

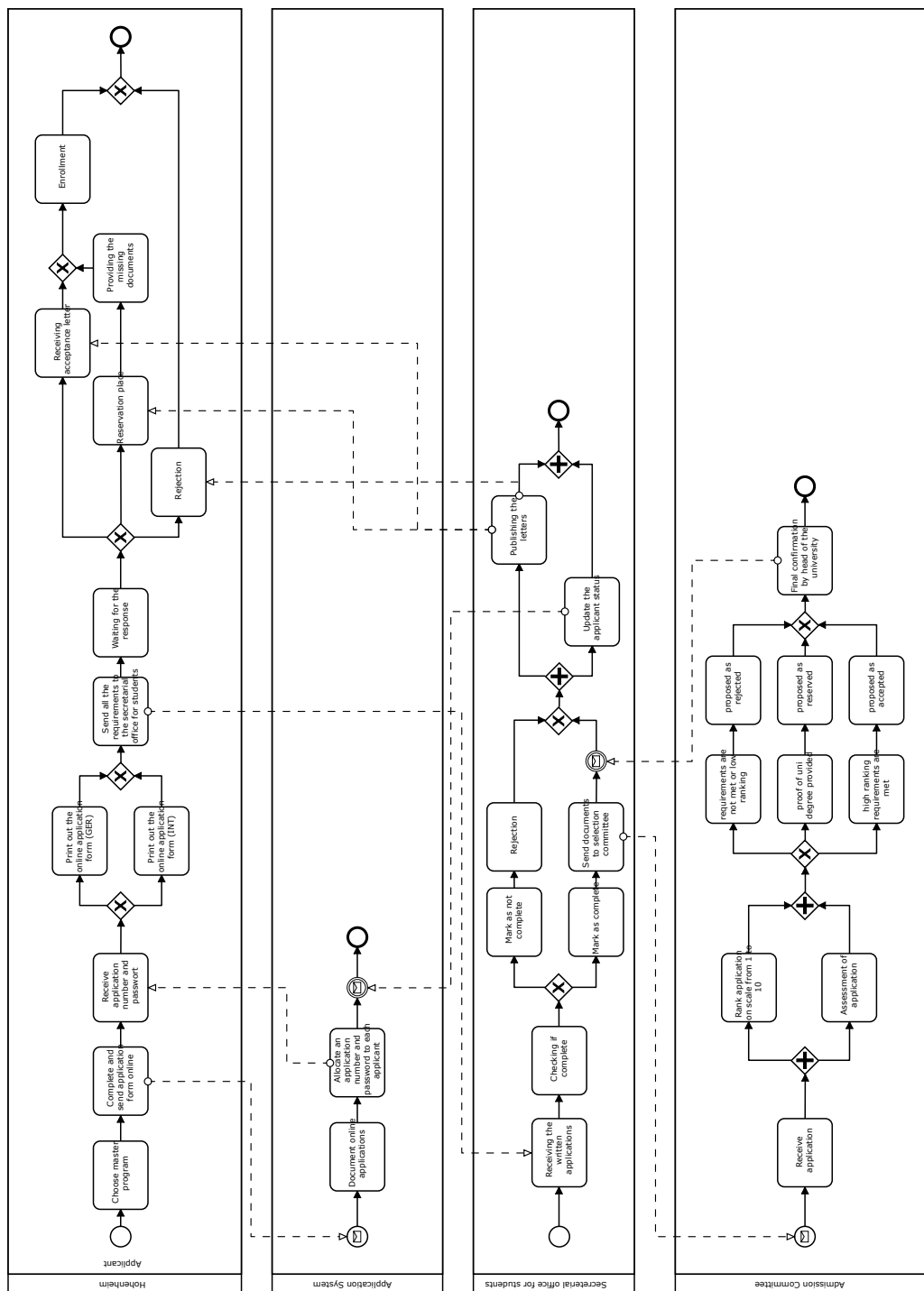


Figure 12 BPMN Hohenheim

Appendix D

Evaluation FCL Solver

This addition contains further evaluation data gathered to assess the FCL Solver. The conduct is described in 4.1.1.

D.1 Second Configuration

Configuration:

- $maxBodyLiteral = 2$
- $pStrict = 50\%$
- $pDefeasible = 20\%$
- $pFact = 30\%$
- $pSuperiority = 5\%$

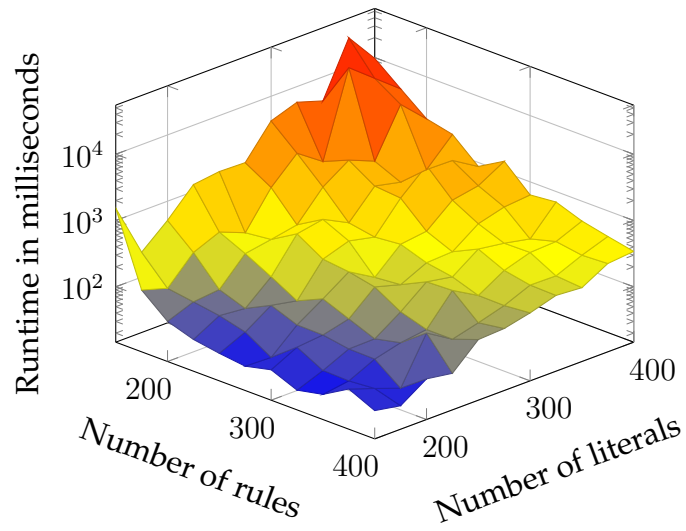


Figure 13 Runtime statistics for the FCL solver (2)

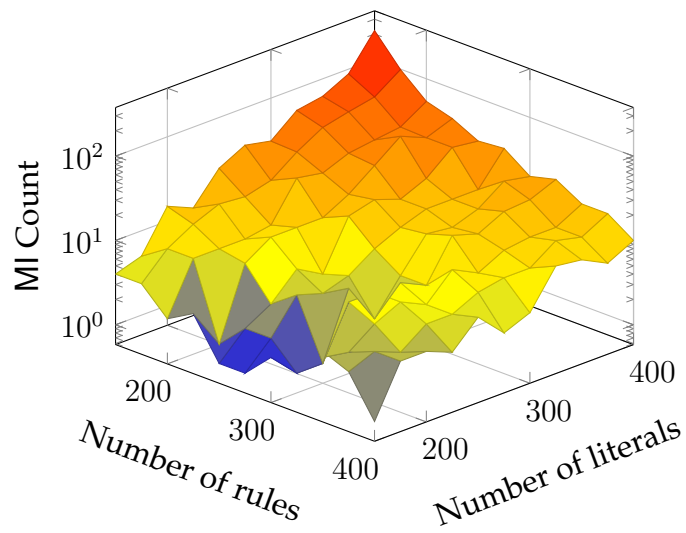


Figure 14 MI in FCL test data 2

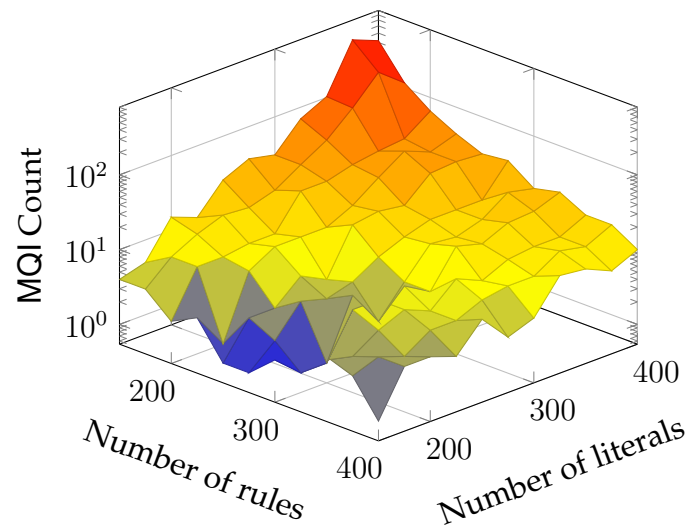


Figure 15 MQI in FCL test data 4

D.2 Third Configuration

Configuration:

- $maxBodyLiteral = 2$
- $pStrict = 20\%$
- $pDefeasible = 30\%$
- $pFact = 50\%$
- $pSuperiority = 5\%$

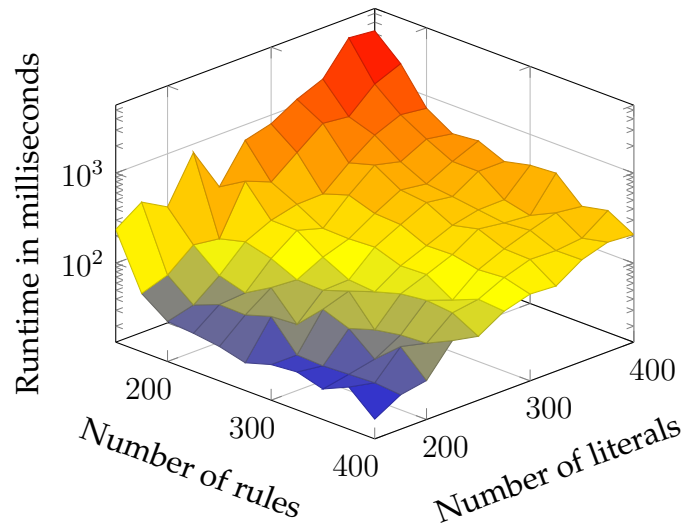


Figure 16 Runtime statistics for the FCL solver (3)

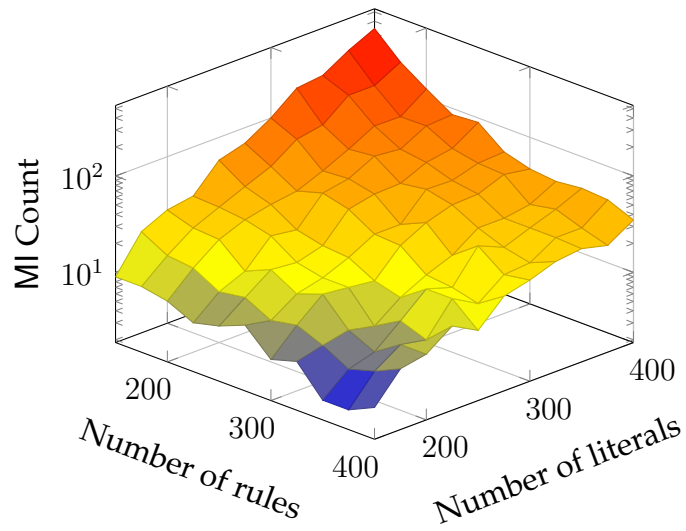


Figure 17 MI in FCL test data 3

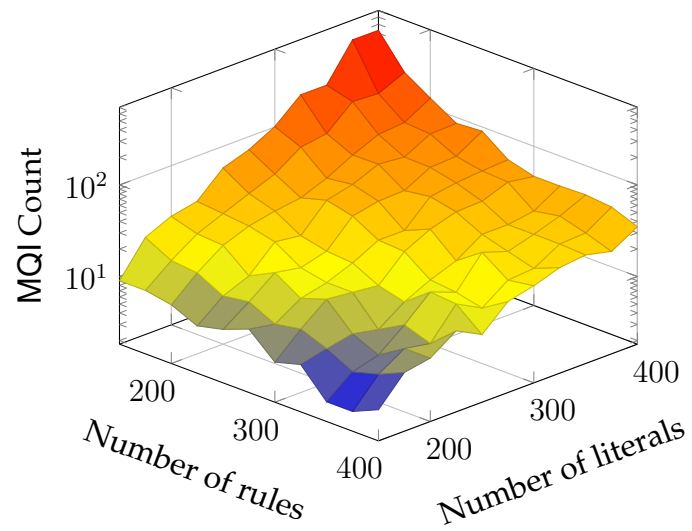


Figure 18 MQI in FCL test data 3

D.3 Fourth Configuration

Configuration:

- $maxBodyLiteral = 3$
- $pStrict = 30\%$
- $pDefeasible = 50\%$
- $pFact = 20\%$
- $pSuperiority = 5\%$

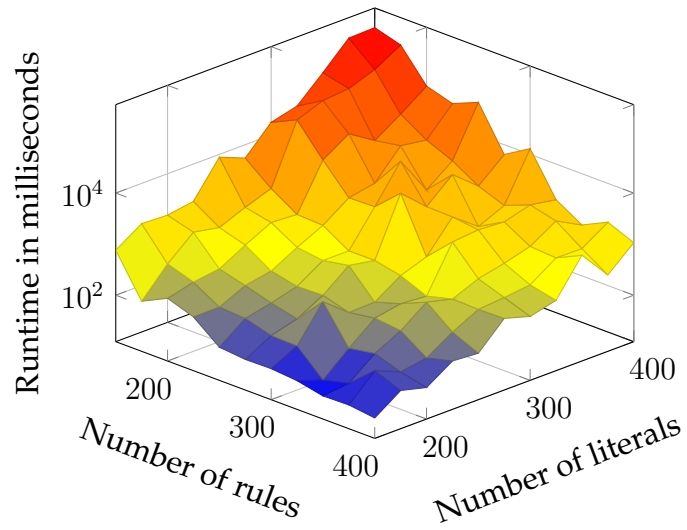


Figure 19 Runtime statistics for the FCL solver (4)

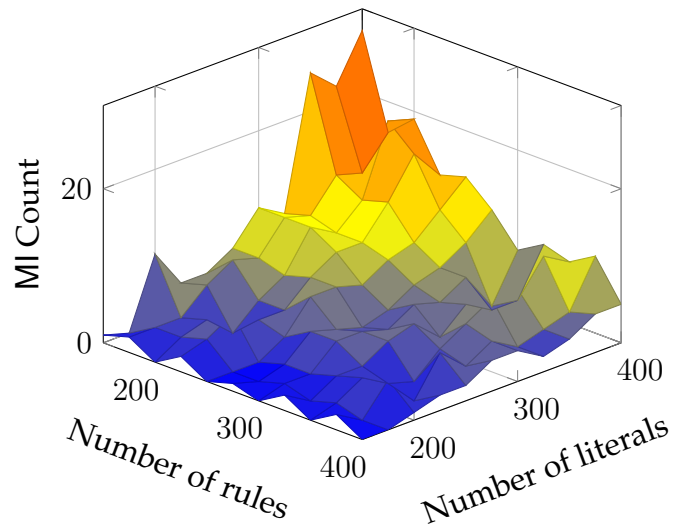


Figure 20 MI in FCL test data 4

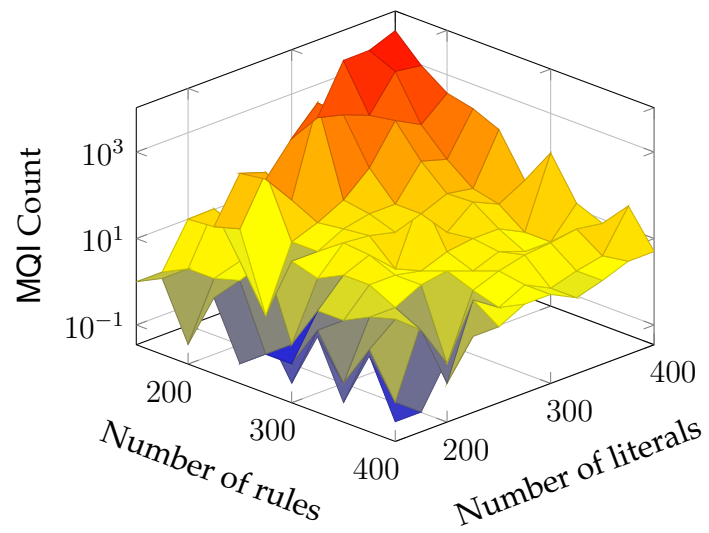


Figure 21 MQI in FCL test data 4