UNIVERSITY OF BREMEN

MASTER'S THESIS

Design and implementation of advanced features for bounded model checking on UML/OCL models

Author:
Jan Prien

Supervisor:
Prof. Dr. Martin Gogolla
Examiner:
Dr. Sabine Kuske

A thesis submitted in fulfilment of the requirements for the degree of Informatics Master of Science (M.Sc.)

March 18, 2020

Declaration of Authorship

Prien	Jan	4136231
Nachname	Vorname	Matrikelnr.

Urheberrechtliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.
Alle Stellen, die ich wörtlich oder sinngemäß aus anderen Werken entnommen habe, habe ich unter Angabe der Quellen als solche kenntlich gemacht.

Datum	Unterschrift

Erklärung zur Veröffentlichung von Abschlussarbeiten

Die Abschlussarbeit wird zwei Jahre nach Studienabschluss dem Archiv der Universität Bremen zur dauerhaften Archivierung angeboten. Archiviert werden:

1. Masterarbeiten mit lokalem oder regionalem Bezug sowie pro Studienfach und Studienjahr 10 % aller Abschlussarbeiten
2. Bachelorarbeiten des jeweils der ersten und letzten Bachelorabschlusses pro Studienfach und Jahr.

☒ Ich bin damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

☐ Ich bin damit einverstanden, dass meine Abschlussarbeit nach frühestens 30 Jahren (gem. §7 Abs. 2 BremArchivG) im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

☐ Ich bin nicht damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

Datum	Unterschrift

UNIVERSITY OF BREMEN

Abstract

Informatics Master of Science (M.Sc.)

Design and implementation of advanced features for bounded model checking on UML/OCL models

by Jan Prien

The latest version of the tool UML-based Specification Environment (USE) supports functionality for model validation and verification (V&V). A plugin provides finding Unified Modeling Language (UML) object diagrams for Unified Modeling Language (UML) class diagrams. The class diagrams can be optionally enriched by invariants. They can be formulated as Object Constraint Language (OCL) constraints. In this paradigm the class diagrams are models. The object diagrams are model instances. A general model validation and verification (V&V) task is to verify that selected properties are fulfilled by the model. Such properties can be added to the model as constraints. While it may be infeasible to validate a constraint for all possible instances of a model (general model checking), there is the alternative to validate the constraint on a selected finite set of instances of the model (bounded model checking). Bounded model checking is applicable in several verification tasks, e.g. validation of model consistency or property satisfiability. Bounds regarding the dimensions of the model come into focus, e.g. minimum and maximum number of objects for a specific class. In USE such bounds are utilised as an input for the instance finder and determine its output, e.g. a set of model instances. In this work, the relevance of additional functionality for working with model bounds for the instance finder is discussed and delivered as prototypes. A concept for comparison of bounds specifications and a bounds tightening mechanism is implemented. With this, a clever bounds specification generation mechanism with various possibilities to define initial bounds is also implemented. The definition of initial bounds is very flexible. Such definitions can be simple or even very complex. For validation of model specifications certain validity rules are defined. A mechanism for validation and computation of applicable proposed bound adjustments is provided. Inappropriate bounds specifications may cause long V&V computation times for instance finding. This stands in the way of an efficient application of the model checking based verification. It is presented here, to what extent the prototypical mechanisms solve this problem.

Contents

List of Figures

List of Tables

List of Abbreviations

CIM Computation Independent Model 8, 9

CLI command-line interface xii, 10, 31, 43, 44, 59, 60, 63, 100

CSP constraint satisfaction problem xii, 2, 3, 17–19, 39–44, 47, 78

DSL domain-specific language 8

GUI graphical user interface 10, 17, 33, 44, 64

ISO International Organization for Standardization 8

MDA model-driven architecture ix, 4, 8, 9

MDD model-driven software development 7–9

MDSE model-driven software engineering 1, 3, 4, 7–10, 12

MOF Meta-Object Facility 4

OCL Object Constraint Language v, 1, 3, 4, 6, 10–14, 21, 31, 33, 40, 43, 44, 59, 64, 78

OMG Object Management Group 4, 6, 8

PIM Platform Independent Model 8, 9

PSM Platform Specific Model 8, 9

SAT satisfiability 10

UI user interface 22, 31, 39, 46, 59

UML Unified Modeling Language v, ix, xi, xii, 1–8, 10–14, 16, 21, 31, 33, 35–37, 42–45, 59–61, 63–65, 67, 77, 78, 100

USE UML-based Specification Environment v, ix, xii, 1–3, 5, 7, 10–13, 15, 16, 21, 31, 33, 35–38, 42–46, 59–61, 63–65, 67, 70, 77, 78, 100

V&V validation and verification v, 1–3, 10, 12, 21, 77

List of Listings

Chapter 1

Introduction and motivation

Model-driven software engineering (MDSE) is state of the art methodology as an approach to create domain models regarding specific problems. Models in form of Unified Modeling Language (UML) class diagrams are widely in use for this purpose. Unsuitabilities regarding the intended use of models can be unwanted aspects of the design. The inclusion of invariants on the UML class diagrams expands the problem of obviation of unsuitabilities. The Object Constraint Language (OCL) is aligned with the latest version of UML. Invariants on UML class diagrams can be formulated with OCL constraints. Additional properties of UML class diagrams can also be formulated as OCL expressions. While it is often almost impossible to determine by simple observation whether a model meets selected requirements, it may even be necessary to formally prove this. The underlying problem can be formulated in such a way that the model must fulfil selected properties. Considering potential model validation and verification (V&V) objectives, there must be a process for proving that selected properties are fulfilled for a model. It should be repeatable while effort and costs remain acceptable.

The tool UML-based Specification Environment (USE) supports modeling functionality. A model consists of an UML class diagram and OCL constraints. UML object diagrams are instances of class diagrams. USE already provides an instance finder. This forms the potential advanced basis for model V&V. A general model V&V task is to verify that selected properties are fulfilled by the model. Such properties can be added to the model as constraints. The instance finder provides computation of finite set of instances for a given model. While it may be incomputable to validate a constraint for all possible instances of a model (model checking), there is the alternative to validate the constraint on a selected finite set of instances of the model (bounded model checking). Bounded model checking is applicable in several verification tasks, e.g. validation of model consistency or property satisfiability. Bounds regarding the dimensions of the model come into focus, e.g. minimum and maximum number of objects for a specific class. USE also supports the model specific specification of these bounds as input for the instance finder. The output of the instance finder is a consequence of the specified bounds. It is a finite set of model instances. Since these bounds specifications may also be unsuitable regarding the intended use, they should be also validated for certain requirements.

Inappropriate bounds specifications often result in long computation times of instance finding. For example, this occurs when there can not exist an instance for a model with a given bounds specification. Computation times are long, especially when the bounds specification represents a huge search space in terms of model checking. Long computation times hold up the modellers and may lead to the application of the verification mechanism being abandoned. In addition, inappropriate bounds specifications can prevent findings of model instances. Trivial inappropriate aspects that hinder findings of instances can currently be specified. The latter

problem often occurs with the current functionality, that creates a new bounds specification for a given model. In addition, it can be a time consuming task to find out the (relevant) differences between two bounds specifications. Also, when having a bounds specification with the lastly described problem and one without any problems.

The relevance of certain (software) features for working with bounds specifications for the instance finder should be pointed out in this thesis. Prototypes of these features should be delivered. The work is based on the research question

> "Is there a communicable way of ensuring validity and suitability of bounds specifications for the USE instance finder of UML class diagrams?"

with the intend to show the validity of the hypothesis

> "There is a communicable way of ensuring validity and suitability of bounds specifications for the USE instance finder of UML class diagrams"

The focus is on the meaning of validity of bounds specifications. This should be defined in this work. The meaning of the term "communicable" is negligible, because generally USE users are experts with similar comprehension of technical details as the developer her-/himself. Implementing a concept for visualising this complex information is already a challenge.

This work complements the working environment used by Gogolla, Hilken, and Doan (2018). It enriches USE with functionality regarding tasks performed there. They presented V&V use cases, applicating the USE instance finder. Similar tasks were performed by Clarisó, González, and Cabot (2019), with a presentation of time expenditures of selected V&V use cases. A concept of improvement is presented, which should reduce the time expenditures of the instance finding process. An adaption of USE suggests itself. Their crucial part is to map instance solver configurations to constraint satisfaction problems (CSPs). The CSPs are enriched with constraints that are derived from the models. Mechanisms for bounds tightening are then applied on the CSPs. This concept should be adapted.

The scientific and conceptual context is presented in chapter 2. Related work is also presented there. In chapter 3 three problems with the latest version of the USE instance finder plugin are presented. The prototypical implementations to solve these problems are also presented in that chapter. In chapter 4 it is evaluated whether one of the implementations brings benefits in terms of the research question. Lastly the results of the work are summarised in chapter 5. In addition, ideas for further work are outlined there.

Chapter 2

Scientific context and related work

Several existing concepts and tools are included in this work. The following sections present the concept of models and selected paradigms they are applicated in. First of all, the general concept of models in the field of software engineering is presented in section 2.1. Then formalisations of models in form of UML class diagrams, OCL and UML object diagrams are outlined. The widely in use paradigm in software engineering called MDSE is then explained in section 2.2. The paradigm underlines the relevance of models in software engineering. The general concept of model V&V is presented afterwards in section 2.3. It complements the MDSE paradigm. The concept of model checking is then presented in section 2.4 as basis for further work. Section 2.5 presents the tool USE. In addition, the role of model checking is outlined. Finally, the concept of CSPs is explained in section 2.6.

2.1 Models

In the following, an overview to the characteristics and meaning of models in the context of this work is given. For other fields than software engineering there may be other interpretations of the term model. There exists a wide range of model definitions, even in this field. (Muller et al., 2012). Models are part of everyone's everyday life. Again and again we abstract and blur out individual features of an This is, e.g., done to describe it by a term. This ability is given to us by nature. However, the choice of unsuitable models is not excluded. (Ludewig and Lichter, 2013, p. 3-5)

Models are descriptive or prescriptive. A descriptive model is, e.g., a blueprint or a specification of software. A prescriptive model is, e.g., a physical model or a documentation of software. The latter can also be used as a descriptive model. This is the case, for example, if the documentation of a software is used, to rebuild it. The classifications descriptive and prescriptive are not mutually exclusive. (Ludewig and Lichter, 2013, p. 5)

Models must have certain characteristics. An existing, planned, or fictitious original must exist (mapping characteristic). A mapping of attributes of the original to attributes of the model also exists. Models do not contain all attributes of the original, but only a subset (shortening characteristic). Attributes that are not mapped are referred to as excluded attributes. Attributes of the model that were not contained in the original are called abundant attributes. The last of the three mandatory characteristics is the pragmatic characteristic. Models must be applicable to replace the original under certain conditions and with regard to certain questions. (Ludewig and Lichter, 2013, p. 5-6)

Various goals drive the usage of models. Models are often used in education, advertising, games, etc., where the originals are not available for ethical or practical reasons. Such models are descriptive. Formal models are also descriptive. They enable to build formal representations of situations or processes to verify hypothesis

on a past, future or conceivable reality. Models are also used for documentation. As representations for situations, persons, processes or objects, they enable the memory of them and the communication about them. Another goal is pursued with explorative models. They are used to assess the consequences of a proposed change in reality that has not yet been decided. (Ludewig and Lichter, 2013, p. 6-9)

Graphs from graph theory are a common way of representing models in software engineering. These are structures consisting of (optionally) annotated nodes and edges. Nodes and edges can be classified by different representation (e.g. by colour, form or annotation). Edges can also be directed.

Models can be expressed not only by graphs and graphics, but also by special languages. It is often possible to map a graphical representation of a model to a corresponding textual representation. While graphical representations are often easier to understand, textual representations also offer certain significant advantages. These types of formalisations are also often used in combination.

Meta models describe the structure of models as models for models. This abstraction is also a formalisation of models. Processing of models can be designed regarding the attributes described by the meta model. The process definition then can be applied on all model instances of the meta model. In special contexts, the terms model and instance can be used instead of meta model and model. Multi level abstraction is also possible. There can be meta models for meta models and so on.

Models are the main artefacts on information system development processes (Giraldo et al., 2018, p. 686) (Florez and Leon, 2018, p. 354). The Object Management Group (OMG) is an organisation that provides meta models for modeling with structure definitions of various model types. With the model-driven architecture (MDA) they also provide a definition of basic principles for working and managing models. As a framework for modeling in object orientated paradigms UML class diagrams in combination with OCL constraints are commonly used. Generally the OMG Meta-Object Facility (MOF) is one of the main frameworks of MDSE (Jácome, Ferreira, and Corral-Diaz, 2017). The MOF includes the UML. These UML class diagrams, OCL constraints and UML object diagrams are presented in the next three sections.

2.1.1 Unified Modeling Language class diagrams

UML is a standard proposed by OMG (*Unified Modeling Language (UML), Version 2.5.1* 2017). The latest version is 2.5.1. A class diagram is a structural UML diagram that models the structure of a system with the elements class, attribute, operation and relationship. These types of elements underlie rules concerning the visual presentation. Several aspects of UML class diagrams are outlined in the following.

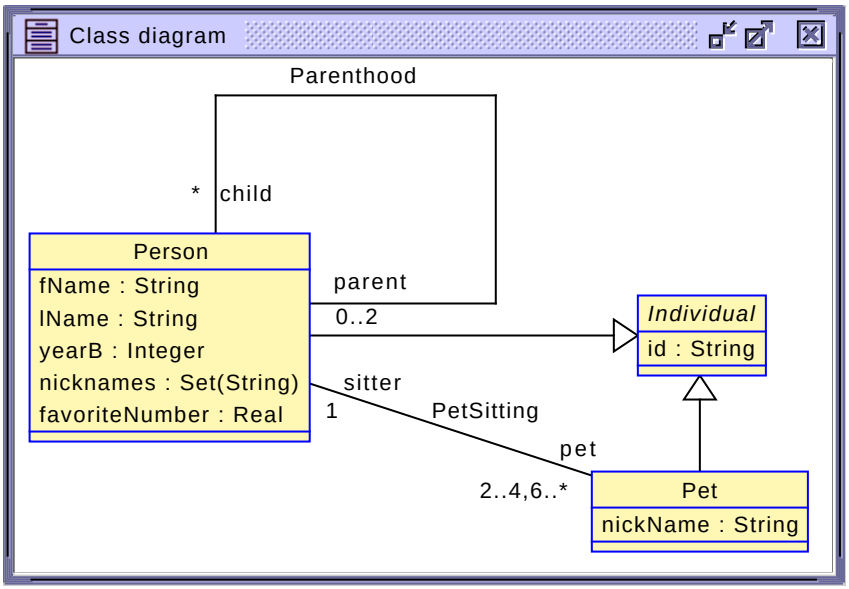

FIGURE 2.1: Unified Modeling Language classes example modeled with UML-based Specification Environment

Figure 2.1 shows the classes with the names `Individual`, `Person` and `Pet`. Classes model the characteristics of objects. Objects are instances of classes. Each class has a name. Here, each rectangle with non-white background represents a class. Those squares are divided in three parts. The top part contains the class name. Class names are unique in the context of a class diagram. Classes can be derived from one class each, so that there exists an inheritance hierarchy. With abstract classes another structuring possibility is provided. Here, abstract classes will be visualised with italic names. The figure shows the abstract class `Individual`. There can not exist objects which have an abstract class as most precise type. Regarding inheritance, classes and abstract classes are treated equally. Classes and abstract classes can be related to abstract classes as implementation of the abstract class. This is a special form of inheritance. The figure shows two implementations as `Person` and `Pet` both implement `Individual`.

Attributes can be modeled on classes. They are contained in the middle part of the class representations. Figure 2.1 shows, for example, for the class `Individual` the attribute `id`. Derived classes implicitly contain all attributes of all classes that are more general according to the inheritance hierarchy. Each attribute has a name. The name is unique in context of the class. It is also unique when taking into account all inherited attributes. The classes `Person` and `Pet` from the figure do not contain explicitly the attribute `id` in their displayed attribute list. They implicitly also have this attribute, because it is inherited from the corresponding class `Individual`. Each attribute underlies a given type. The figure shows among others for the attribute `id` the type `String`. The set of types depends on the concrete framework, implementation or tool.

Next to the inheritance there are other relationships that are modeled on classes. Associations are the general form for special relationship types. Each association has a name. The name is unique in the context of a class diagram. Figure 2.1 shows the associations `PetSitting` and `Parenthood`. An association has at least two ends. Each association end links a class while roles are assigned to objects in context of the association. The figure shows the association `PetSitting` with one end for class

`Person` with the role `sitter` and one end for class `Pet` with the role `pet`. Multiple ends of an association can be linked to the same class. The figure shows the association `Parenthood` with two ends for class `Person`. Role names are unique in context of an association. Instances of associations are called links. Associations are meant for the classes level and the corresponding links are for the objects level. For each association end a multiplicity is necessary.

Multiplicities are defined via positive integer ranges with the exception that the upper limits can also be given as unlimited. This is expressed by `*`. A multiplicity is at least one integer range but can also be a set of integer ranges. Figure 2.1 shows the association `PetSitting` with the end `sitter` for class `Person` with the multiplicity `1`. This is just a simplified visualisation for the range `1..1`. It means, that each object of class `Pet` is linked with a link for that association to exactly one object of class `Person`. The multiplicity on the other end means, that each object of class `Person` is linked with that association to exactly two up to four or six or more than six objects of class `Pet`. The figure shows also the multiplicity `*` for role `child` in association `Parenthood` which is just the simplified visualisation for `0..*`.

Not all features of UML class diagrams are mentioned here. All relevant aspects regarding other parts of this work are outlined.

2.1.2 Object Constraint Language invariants

OCL is a standard proposed by OMG (*Object Constraint Language 2.4* 2014). The latest version is 2.4. It is aligned with UML 2.5.1. A shallow explanation of OCL in context of UML class diagrams is given in the following.

OCL allows to define constraints in UML class diagrams. Those definitions can be formulated textually. OCL describes a concrete syntax for textual formulations.

Constraints can be on the one hand pre and post conditions for operations and on the other hand invariants for classes. The former is not relevant regarding this work. Both need to be defined in a specific context. For pre and post conditions this would be a specific method of a specific class. For invariants it would be a specific class. Each constraint needs to be specified in such a context. On model instances it is possible to evaluate whether constraints hold or not.

An exemplary invariant for fig. 2.1 may be that for all instances of class `Individual` the value of the attribute `id` must be unique. Another invariant may be that there must exists at least one instance of class `Person`, that is linked, according to the association `PetSitting`, with at least ten instances of class `Pet`.

In OCL some constraints can be expressed in multiple forms. Not all features of OCL are mentioned here. All relevant aspects regarding other parts of this work are outlined.

2.1.3 Unified Modeling Language object diagrams

An object diagram is a structural UML diagram that models the state of a system with the elements object and link. The systems are modeled with UML class diagrams. Class diagrams are meta models for object diagrams. The types of elements of object diagrams underlie rules concerning the visual presentation. This is similar to class diagrams.

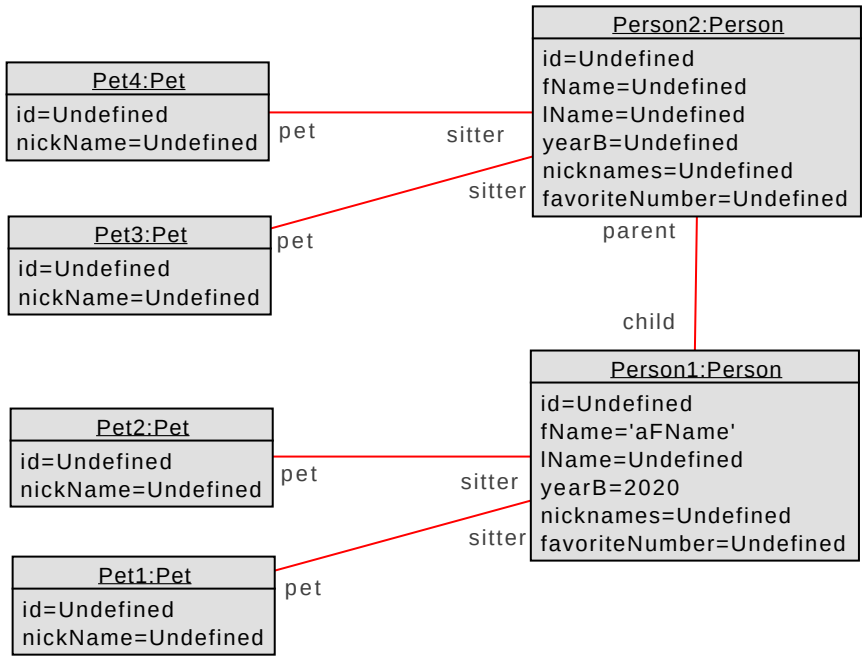

FIGURE 2.2: Unified Modeling Language objects diagram example modeled with UML-based Specification Environment

Figure 2.2 shows an object diagram for the class diagram from fig. 2.1. It contains two objects for class `Person` and four objects for class `Pet`. Each object is represented by a rectangle with non-white background. The squares are divided into two parts. The first part shows the identifier and the class of an object. An object identifier is unique in the context of each object diagram. One link for association `Parenthood` and four links for association `PetSitting` are contained.

2.2 Model-driven software engineering

MDSE is widespread in use (Whittle, Hutchinson, and Rouncefield, 2014). As a paradigm of software engineering, it is "Any activity related to the production or modification of software pursuing some goal(s) beyond the software itself" (Ludewig, 2000). As optimal software engineering is "(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1)" ("IEEE Standard Glossary of Software Engineering Terminology" 1990, p. 67).

Models are the crucial part of MDSE. The form of models varies widely. In the field of software engineering there are two types of models. On the one hand there are software models. The set of possible forms of software models includes, for example, models formulated in natural language, entity-relationship models or use case diagrams. On the other hand there are models for procedures and processes. This includes for example the waterfall model and the V-model. (Ludewig and Lichter, 2013, p. 11)

Model-driven software development (MDD) is the automatic generation of executable software from formal models. This is related to MDSE. Activities regarding transformation of models to executable software are a subset of activities of MDSE.

This is not necessarily an automated process. Transformations like this can also be done by hand. There must be benefits unrelated to MDD that drive MDSE, as Whittle, Hutchinson, and Rouncefield (2014) worked out. It is commonly in use also without the usage of MDD. Selic (2006, p. 609) states that it is driven especially by the potential of the combination of abstraction and automation.

Domain-specific languages (DSLs) are an alternative for common modeling languages. It is not uncommon to develop small DSLs. Even generators for DSLs are developed sometimes. A challenge with this is to integrate multiple DSLs. (Whittle, Hutchinson, and Rouncefield, 2014)

The UML is widely in use. It provides multiple types of models. In 2005 UML was also published as an approved International Organization for Standardization (ISO) standard (*ISO/IEC 19501:2005* 2005).

One of the main modeling tasks is to define models as abstractions of systems. UML class diagrams are commonly used for this. They are used for prescriptive and also descriptive purposes in this context. There are already several tools and automations that support working with such models. Functionalities for code generation from UML class diagrams or deriving UML class diagrams from existing code are used by many people involved in software engineering processes.

The OMG, creator and maintainer of UML, also describes a specific form of MDSE with the MDA. The MDA describes the link between model, platform and code-transformation. It is a reference conceptual framework adopted in many organisations. For automated transformations, the model needs to be specified in a formal language. This is often done using UML. In additional, the transformation steps needs to be specified in a formal language. A meta model for the target model must exist. The target model can be the generated code. This can also be considered as a model. Therefor, this procedure differentiates between source and target model. Generated code mostly can be manually modified in a prescribed manner. This depends on the concrete framework. MDA describes four types of models:

- A Computation Independent Model (CIM) describes software at the business level. It must be formulated in a language that the user understands. The CIM thus determines what the software should achieve.
- A Platform Independent Model (PIM) represents the business functionality of a program or component independently of the details of a particular technical platform, i.e. it abstracts from it. It is written in a formal modeling language that has precise semantics and allows the business functionality to be adequately expressed.
- A Platform Specific Model (PSM) is created by transforming a PIM. It implements the PIM using the specific properties of the specific platform.
- A platform model describes a platform. Platform models are to specify transformations.

The first three types are levels of abstraction of the source model. Platforms can be organised hierarchical. The terms platform independent and platform specific are relative and can be interpreted here as roles for models. Models can have both roles at the same time.

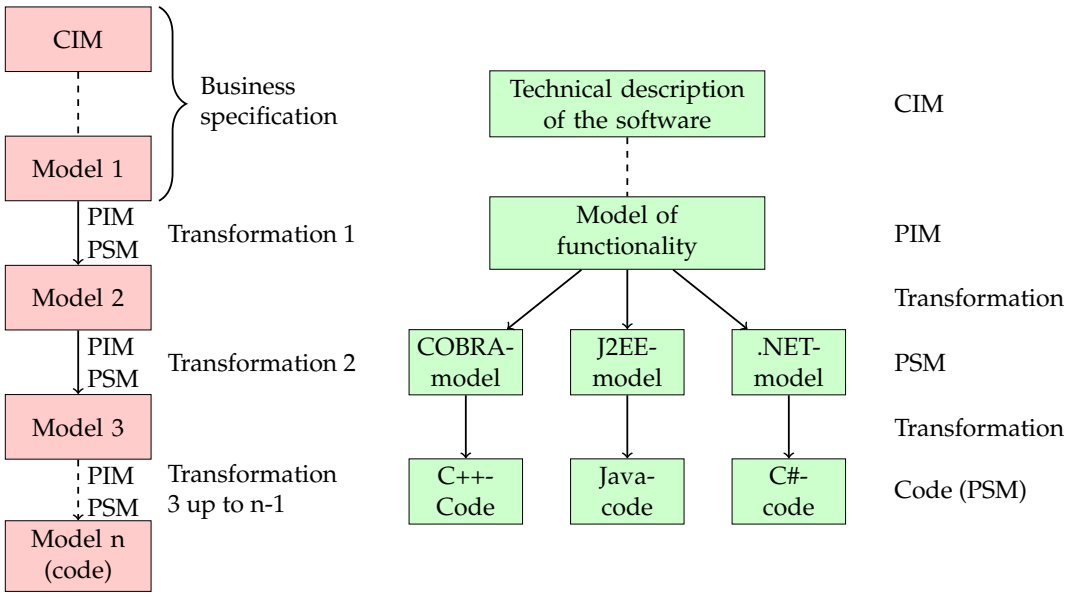

FIGURE 2.3: Model-driven architecture basic principle and example (Adopted from Ludewig and Lichter, 2013, p. 342)

Figure 2.3 shows the basic principle of MDA on the left and an example on the right. Code for three different platforms is generated from one model in two transformation steps. Exemplarily, the J2EE-model is platform independent from the viewpoint of the corresponding PSM and platform specific from the viewpoint of the corresponding PIM. (Ludewig and Lichter, 2013, p. 341-342) (Brambilla, Cabot, and Wimmer, 2012, p. 39-41)

2.3 Model validation and verification

When working on products on an abstract level it may be necessary to verify whether the product satisfies specified requirements or to validate whether the product is applicable for specified use cases under specific circumstances. Identifying unwanted or unsuitable properties early on an abstract product eventually prevents more costly measures later on. Especially in a paradigm like MDSE, in which often abstract models are iteratively conretised, such checks are of great importance. Developing, implementing and applying methodology, frameworks, processes, mechanisms and tools for validation and verification can be very costly. From another viewpoint, such effort can have also a very high value. For example when it is accomplished for a formal modeling framework that is widely is use.

In verification of models there are general difficulties. When verifying whether a model is good, complete or consistent, it is always questioned whether enough properties are specified to check that. There can also be different causes of a model failing to satisfy a property. The model or the properties may be wrong. One may be challenged with determining such causes. (Seshia, Sharygina, and Tripakis, 2018, p. 85)

Cabot, Clarisó, and Riera (2014) state two main problems with inconsistent models in MDD. Both are related to development costs and software quality. Firstly, errors may be detected earlier in the development process. Fixing the errors, detected earlier, oftenly is less costly than on a later stage. Secondly, inconsistencies may directly propagate in implementation errors. Referring to the first problem, solely the

detection at the implementation level may be more expensive than on the design level.

2.4 Bounded model checking

Model checking is a computer-assisted method for the analysis of dynamical systems that can be modeled by state-transition systems (Clarke, Henzinger, and Veith, 2018). The term refers primarily to systems in the form of code. The ideas of model checking and bounded model checking are presented in the following.

Conventionally model checking means to transform the system under investigation into a state transition graph (a.k.a Kripke Structure). The properties to check are formulated in temporal logic. The system fulfils a property if it is valid in all states on all paths, i.e. sequences of states that are reached via state transitions and start with one of the start states of the system. A system does not fulfil a property if the property is not valid in at least one state on one such path. Because of the possibility to specify systems that express infinite paths this decision is potentially incomputable. This is a relevant problem, despite of existing special designs, which cannot prevent the problem. For a terminating decision procedure not all paths of the system can be taken into account. Bounded model checking means using specific bounds that limit the paths to be considered to a finite amount, while the paths are also finite. The result of bounded model checking is not directly equate to the result of general model checking. It can have an equate value if an additional logical justification is given, stating that all the ignored paths are irrelevant for the verification of the specific properties. (Clarke, Henzinger, and Veith, 2018) (Biere and Kröning, 2018)

Abstracting from the concept behind the common interpretation of the term model checking, it can be interpreted as taking a formal model and a set of properties and compute which instances of the model satisfy and not satisfy which properties. Since there can be models with infinite numbers of instances this is also potentially incomputable. So bounded model checking can be interpreted as using a finite set of model instances for the decision. This represents a terminating and therefor applicable verification method, whose results may already satisfy the needs of the user.

The underlying technology for the decision process is undetermined. Modern satisfiability (SAT) solvers have become the core technology of many model checkers (Biere and Kröning, 2018).

2.5 USE Tool

USE is a system for the specification and validation of information systems based on a subset of the UML and the OCL (*USE: UML-based Specification Environment* 2020). The latest version is 5.1.0. An overview to selected characteristics of USE and its role in MDSE and model V&V is given in the following.

USE provides functionality for the design of UML/OCL models. UML/OCL models can be stored and read from files in form of textual specifications. For loading and working on UML/OCL models there are multiple user interfaces, a graphical user interface (GUI) and a command-line interface (CLI). Both provide the functionality that is outlined in the following. The CLI is also integrated in the GUI. Only the GUI enables visualisation of UML class and object diagrams.

An UML object diagram can be edited with textual commands on the CLI. A system must be loaded for this. A system is an UML class diagram that can be

enriched with OCL constraints as invariants. Invariants have an unique name in context of an UML class diagram. A special syntax is provided for textual commands in the so called USE Shell. The concrete syntax is not relevant in this work, but the general functionalities will be outlined. Objects for classes can be created and destroyed. The unique names of objects are used as textual references for them. The values for attributes can be set and also overwritten for specific objects. Links for associations can be inserted for existing objects. Existing links can get deleted. OCL expressions can be evaluated. This can be done by using all or specific invariants or by directly entering OCL expressions. There are also more USE Shell commands provided (*USE documentation* 2007) but all the actions that are relevant for this work are already outlined.

Supported UML class diagram attribute types are individual for tools, frameworks and so on. USE supports various types. It is differed between simple types and collection types. Simple types are Integer, Real, Boolean, String or classes. Collection types are Set, Sequence, Bag or enumerations. A Set is an unordered collection of items, which do not contain duplicates. A Sequence is an ordered collection of items, that may contain duplicates. A Bag is an unordered collection of items, that may contain duplicates. There is no direct way to restrict types in collections. Items in collections can have each of the presented types.

Generally USE supports not all UML and OCL features. As Gogolla, Hilken, and Doan (2018) summarised, certain features regarding operations are not supported. For OCL, the only fully supported collection type is Set and the fully supported simple types are Integer, Boolean and classes. Certain OCL operations, like “substring” and “concat”, are also not supported. The powerful operation “iterate” is also not supported. (Gogolla, Hilken, and Doan, 2018, p. 26)

```
1  model ClassDiagramExample
2
3  abstract class Individual
4  attributes
5    id:String
6  end
7
8  class Person < Individual
9  attributes
10   fName:String
11   lName:String
12   yearB:Integer
13   nicknames:Set(String)
14   favoriteNumber:Real
15 end
16
17 class Pet < Individual
18 attributes
19   nickName:String
20 end
21
22 enum AnimalType { FarmAnimal, DomesticAnimal, OtherAnimal }
23
24 association Parenthood between
25   Person [0..2] role parent
26   Person [0..*] role child
27 end
28
29 association PetSitting between
30   Person [1] role sitter
31   Pet [2..4,6..*] role pet
```

```
32 end
```

LISTING 2.1: UML-based Specification Environment specification for Unified Modeling Language classes example from fig. 2.1

Listing 2.1 shows the USE specification for the UML class diagram in fig. 2.1. The specification is bound to a concrete syntax for USE model specifications. UML class diagram and OCL invariants can be specified textually that way. In addition to the interpretation from section 2.1.1, firstly a specification defines the name of the model with the keyword `model`. The first line contains the keyword following the name. The name is `ClassDiagramExample` here.

```
1 !create Person1,Person2 : Person
2 !create Pet1,Pet2,Pet3,Pet4 : Pet
3 !set Person1.fName := 'aFName'
4 !set Person1.yearB := 2020
5 !insert (Person2,Person1) into Parenthood
6 !insert (Person1,Pet1) into PetSitting
7 !insert (Person1,Pet2) into PetSitting
8 !insert (Person2,Pet3) into PetSitting
9 !insert (Person2,Pet4) into PetSitting
```

LISTING 2.2: UML-based Specification Environment commands for Unified Modeling Language objects diagram example from fig. 2.2

As fig. 2.2 shows an object diagram for fig. 2.1/listing 2.1, listing 2.2 shows the USE Shell commands that constructs the object diagram, when interpreted top down line wise.

USE already provides several functionality for model V&V with the outlined characteristics. A framework for manual inspection is given with the manipulation of UML object diagrams and the evaluation of OCL expressions in those models. Test cases implicitly can be specified formally, since those models can be saved in recoverable form.

Beside the V&V advantages, USE generally enhances MDSE. The textual specification is optimal for collaborated work using common version control systems. The corresponding syntaxes are a formalisation that also supports custom processing.

For V&V USE also provides extended functionality. Properties of UML class diagrams can be evaluated with bounded model checking.

2.5.1 Model instance finding with bounded model checking

A plugin for USE is given with the USE Model Validator. It supports finding UML object diagrams for UML class diagrams, respecting OCL invariants and OCL constraints. The process for finding of UML object diagrams is explained in the following and is linked with the concept of bounded model checking. Several use cases and possibilities to utilise the bounded model checking concept are presented.

First of all, the concept of bounded model checking includes the bounds regarding the dimensions of the model. Bounds regarding the model elements class and association and domains for types of attributes are crucial for UML class diagrams. Understanding the specification of bounds as a bounds model, also a meta model for this must exist as formalised basis for automised/formalised processing. While a bounds meta model can be very complex and such bounds specifications must satisfy selected requirements, not the most precise bounds meta model may be the best. For example, complexity and understandability may need to be combined. There are

atomary dimensions, e.g. class specific minimum number of objects or attribute for class specific type domain bounds. Dimensions can also be be combined, e.g. general type specific domain bounds that are applied for all attributes with the type. The concrete UML class diagram bounds meta model supported by USE is presented in the next section.

With a set of object diagrams, that is implicitly defined by a class diagram bounds model, bounded model checking can be performed. In terms of bounded model checking the set of object diagrams represents the set of model instances. The properties they must satisfy are additional OCL constraints. Therefor, bounded model checking can also be applied as evaluating whether, in abstract, all model instances satisfy all properties or, in precise, all object diagrams satisfy all OCL constraints. A property is not fulfilled in terms of bounded model checking if there exists at least one object diagram that does not satisfy at least one constraint. A property is fulfilled if all object diagrams satisfy all constraints. Adding additional meta level, this can be utilised in certain use cases. On the below level, USE supports the instance finder, which takes an UML class diagram, OCL invariants and a bounds specification and computes a finite set of UML object diagrams, where each object diagram satisfies all invariants. On the meta level the set of UML object diagrams can be interpreted or applied in the use cases presented in the following.

Gogolla, Hilken, and Doan (2018) presented eight use cases for utilising the instance finder with its results. The interpretations of no found instances in the following are not generally valid. The derived statements are always weakened by the fact that also the inappropriate search space could have been caused the result.

1. Consistency of a model can be verified by finding instances with the instance finder. A UML class diagram, OCL invariants and a bounds specification are used as input. If at least one instance can be found, "the class model including the model-inherent multiplicity and whole-part constraints and the explicit OCL invariants are not contradictory" (Gogolla, Hilken, and Doan, 2018). No found models can be interpreted as the opposite. Przigoda, Wille, and Drechsler (2018, p. 84) differentiate between weak and strong consistent models. A model is weak consistent if there exists an instance at least for every class. A model is strong consistent if there exists an instance at least for every nonabstract class. UML implicitly supports checking of both types. But no explicit differentiation is implemented.

2. Property satisfiability can also be verified by finding instances with the instance finder. This is already included in the first use case. Additional input is the property in form of an OCL constraint. Here, also the opposite can be shown with no found instances.

3. Implication of additional constraints can be verified by adding the negated OCL constraint to the model. If the instance finder finds at least one instance, the (unnegated) constraint is not implied by the original model. No found instances means that the constraint is implied.

4. Constraint independence can be verified by removing the constraint from the model and applying the constraint implication concept from the third use case for the removed constraint. This means removing the OCL constraint and adding its negation. If the constraint is implied by the model without the constraint itself, it is not independent. When the constraint is not implied, it is not independent.

5. Although manual inspection was presented as conditionally insufficient, it sometimes best serves the intended purpose. As solution interval exploration, the advanced utilisation of the set of found model instances is meant. Important observations can be made on this basis, for example.

6. Assuming a partial described model instance that might not already satisfy model inherent or explicit constraints, finding a possible completion or all possible completions can be the desired purpose. The result, a set of model instances, can be used again as basis for further processing.

7. For verification of constraint equivalence implication, assuming two constraints `C1` and `C2`, the logically negated invariant `(C1 implies C2) and (C2 implies C1)` can be used as additional input beside the UML class diagram, OCL invariants and a bounds specification. If the instance finder finds at least one instance with this input, the constraints are not equivalent in context of the given model. No found instances means that the constraints are equivalent.

8. Another use case arises with utilising the set of found model instances, found with an UML class diagram, OCL invariants and a bounds specification as input, and assuming classifying terms in form of OCL constraints. These inputs can be used to inspect one or all model instances that belong to a selected group of the groups implied by the given classifying terms. Interpreting the finite number of groups as equivalence classes, a concrete use case may be to manually inspect representatives for all equivalence classes.

2.5.2 Instance finder configurations

Instance finder configurations represent bounds for bounded model checking. They can be saved in recoverable form. The details of instance finder configurations are outlined in the following.

There is an implicit model of instance finder configurations. Each configuration has a name. For all classes, the minimum and maximum number of objects must be defined. A set of preferred instance names can also be specified. For each attribute of each class, the minimum and maximum number of defined attributes on objects must be specified. This can also be specified as obligatory. Then for each object of the class the attribute must be defined. It can also be specified as that the maximum is unlimited. For attributes with collection type, the minimum and maximum number of items must be specified. The maximum can be set as unlimited. A set of preferred values can be specified for each attribute. It is not supported to specify bounds of inherited attributes. Attribute bounds can only be specified on the most general class having the attribute. For each association, the minimum and maximum number of links must be defined. The maximum can be set unlimited. A set of required values can also be specified by using tuples of object names. For the simple types Integer, String and Real also bounds are necessary. For the type Integer, the minimum and maximum value must be specified. A set of required values can also be specified. For the type String, the minimum and maximum number of different values must be specified. A set of preferred values can also be specified. For the type Real, the minimum and maximum value and the step width must be specified. The specification of the step width makes the set of Real values finite. A set of required values can also be specified. For each invariant, it must be specified, whether the invariant is active and whether it is negated. Additionally some options can be set. It can be set, whether no cycles are allowed inside of aggregations and compositions. It

can also be set, whether no objects participating in a composition can be member of more than one composition.

There is no complete specification of valid domains and it is possible to specify illogical bounds. For this reason, no concrete domains are presented here. Relevant bounds domains are of two types. Real number domains (for Real type settings) and integer domains (for all other bounds) are included. There are also derived bounds domains. For example, the instance names used in the specification of required links can also be interpreted as required instance names, which are bound to the maximum number of instances of the corresponding class. Those required instance names effect the usage of specified preferred instance names. They may become obsolete because of the implicit specified required instance names.

Not all aspects of the configuration must be explicitly defined, when loading an instance finder configuration from a file. A default value will be used instead. There exists no complete specification of default behaviour for not explicitly specified bounds.

Configurations can be expressed in Java Properties format. This is a list of key value pairs. The keys depend on the model elements names for classes, attributes, associations and invariants. In Java Properties files a line contains either nothing, a comment or a key value pair optionally followed by a comment. USE only supports extended Java Properties, which additionally contain sections. The key value pairs are not interpreted as one set of key value pairs, but as one or more sets of key value pairs. The Java Properties file therefor contains section names that identify the sets of sets.

```
1  [config1]
2
3  Integer_min = -10
4  Integer_max = 10
5
6
7  # ------------------------------------------------------------------------
       Individual
8
9  Individual_id_min = -1
10 Individual_id_max = -1
11
12 #
       ------------------------------------------------------------------------
        Person
13 Person_min = 1
14 Person_max = 1
15
16 Person_fName_min = -1
17 Person_fName_max = -1
18 Person_favoriteNumber_min = -1
19 Person_favoriteNumber_max = -1
20 Person_lName_min = -1
21 Person_lName_max = -1
22 Person_nicknames_min = -1
23 Person_nicknames_max = -1
24 Person_nicknames_minSize = 0
25 Person_nicknames_maxSize = -1
26 Person_yearB_min = -1
27 Person_yearB_max = -1
28
29 # Parenthood (parent:Person, child:Person) - - - - - - - - - - - - - - - -
       - - - - -
30 Parenthood_min = 1
```

```
31 Parenthood_max = 1
32
33 # PetSitting (sitter:Person, pet:Pet) - - - - - - - - - - - - - - - - - -
     - - - - -
34 PetSitting_min = 1
35 PetSitting_max = 1
36
37 #
     ------------------------------------------------------------------------
      Pet
38 Pet_min = 1
39 Pet_max = 1
40
41 Pet_nickName_min = -1
42 Pet_nickName_max = -1
43 #
     ------------------------------------------------------------------------

44 aggregationcyclefreeness = on
45 forbiddensharing = on
46
47 [config2]
48
49 Pet_max = 10
50 Person_max = 10
51 PetSitting_max = -1
```

LISTING 2.3: UML-based Specification Environment textual instance finder configurations example for model from fig. 2.1

Listing 2.3 shows the content of an instance finder configuration file. It contains the two configurations `config1` and `config2`. Each starts from the line containing the name in rectangle brackets and ends with the start of a new section or the end of the file.

`config1` from listing 2.3 contains keys of a default configuration. Each model elements name can be found here, e.g. names of classes, attributes (specific for class) and associations. For invariants the names would be used as partial keys. In the following `<name>` is a placeholder for the elements name. For invariants the key `<name>_negate` and `<name>_active` can be used. For the types Real, Integer and String the keys `<name>_min` and `<name>_max` specify the lower and upper bounds. `<name>` specifies the preferred values for Real and Integer and the obligatory values for String. For type Real, `<name>_step` specifies the step width. The settings for the type are interpreted as disabled if no key is given for a whole type. For classes, but not for enumerations, the keys `<name>_min` and `<name>_max` specify the minimum and maximum number of objects. `<name>` specifies the preferred names for objects. In the following `<cName>` is a placeholder for the name of the class, which owns an attribute. `<aName>` is a placeholder for the name of the attribute. The minimum and maximum number of defined attributes is specified by `<cName>_<aName>_min` and `<cName>_<aName>_max`. `-1` for `<cName>_<aName>_min` specifies obligatory definition of the attribute for all objects of the class. In this case `<cName>_<aName>_max` becomes irrelevant. `-1` for `<cName>_<aName>_max` specifies upwardly unlimited defined attributes. `<cName>_<aName>` specifies the preferred values for attributes. For collection type attributes the keys `<cName>_<aName>_minSize` and `<cName>_<aName>_maxSize` specify the minimum and maximum number of contained items. `-1` for `<cName>_<aName>_maxSize` specifies upwardly unlimited numbers of items. The minimum and maximum number of links for associations is specified by `<name>_min` and `<name>_max`. `-1` for `<name>_max` specifies upwardly

unlimited links. <name> specifies the obligatory links. aggregationcyclefreeness specifies whether no cycles are allowed inside of aggregations and compositions. forbiddensharing specifies whether no objects participating in a composition can be member of more than one composition. Values for both can be either on or off. config2 from listing 2.3 contains only selected keys with exemplary values. The order of the keys is irrelevant. The value of the latter occurrence will be read, when one key is used more than once in a configuration section.

The minimum and maximum number of objects for abstract classes can be specified in the file. In the GUI they are not displayed but correctly loaded. Both sometimes are somehow set to 1 when other modifications on the configuration are processed. This is also not visible for the user. Some other properties values may not be valid and are replaced by default or derived values when loaded. This is ignored here, because there is no complete specification of this behaviour.

2.6 Constraint satisfaction problems

CSPs mostly can be used instead of the original formal representation applicated in model checking. Several concrete solving methods are already defined and implemented for CSPs. In the following, the definitions of CSPs are given.

For CSPs, the concepts of variables, values for variables, domains for variables and constraints are crucial. A value is something that can be assigned to a variable. Values and variables are of a specific type. For example, there are numeric variables. Also, sub types of such types exists. For example, integer or real number variables are both numeric. Whether a value can be assigned to a variable depends on the value. It must have a type that is the same or a sub type from the type of the variable. A domain of a variable is the set of values that the variable can take. Constraints are defined on a non empty set of variables. They represent a restriction on the set of values that these variables can take simultaneously. A simultaneously assignment of values to a set of variables is an instantiation. An instantiation is total if it respects all variables of the problem. Otherwise, it is a partial instantiation. Total or partial instantiations can be consistent or inconsistent. A consistent instantiation must satisfy all constraints. When not all constraints are satisfied, the instantiation is inconsistent. A partial instantiation that can not be extended to a total consistent instantiation is called a Nogood. (Ghedira, 2013, p.1-5)

A CSP is a triple (V, D, C) with $V = \{v_0, \ldots, v_n\}$ as a finite set of n variables, $D = \{d_0, \ldots, d_n\}$ as a finite set of n finite and discreet domains and C as a finite set of m constraints. Each constraint involves a subset of k variables from V such as $k \geq 1$. Each variable $v_i \in V$ has a domain $d_i \in D$.

The solution of a CSP is a consistent total instantiation. A CSP is consistent if it has a least one solution. There are several possible representations for CSPs. Predicates for the variables and optionally for domains are needed for textual representation. CSPs can be textually visualised with the mathematical definition from above and with referencing variables in constraints by using their representations. Graphs can be used as graphical representation. Nodes represent the variables and the edges represent the constraints. The nodes contain nodes for each (relevant) value of their domain. (Ghedira, 2013, p.4-5)

Further definitions of or on CSPs are not relevant here, but provide an extended basis for solving methods.

```
1 V = {
2     Person ,
```

```
3     Pet,
4     Individual
5   }
6
7   D = {
8     {1,2},
9     {1,2,3,4,5,6,7,8,9,10},
10    {1,2,3,4}
11  }
12
13  C = {
14    "Individual= Person+ Pet",
15    "Pet≥ 2 ∗ Person"
16  }
```

LISTING 2.4: Constraint satisfaction problem (CSP) example in textual form

Listing 2.4 shows an exemplary CSP in textual representation. The CSP has, among other things, the variable `Person` with an integer domain containing the numbers `1` and `2`. Overall the CSP has three variables and two constraints. The first constraint `Individual= Person+ Pet` involves all variables and represents that `Individual` must be equal to the sum of `Person` and `Pet`. This CSP has multiple solutions, e.g. `(Person→ 1, Pet→ 2, Individual→ 3)`. Both constraints are satisfied in the solution. There is for example no solution consisting of the partial instantiation `(Individual→ 2)`. When `Individual` has the value `2`, the first constraint implies that `Pet` and `Person` must be `1`, considering the domains of these variables. But the second constraint can not be satisfied. This example contains several partial instantiations that can not be part of any solution.

For solving CSPs efficiently, search space reducing comes in mind (Cooper, Jguirim, and Cohen, 2018, p. 65). Reducing the search space means not to change the set of possible solutions, but avoid exploring search space that can never be part of a solution. The search space is defined by the domains of the variables or, more precisely, by the domains of variables that are part of at least one constraint. A minimal search space can be achieved by eliminating all domain elements which can not be contained in any solution. In other words, this means to remove all partial instantiations which can not be part of any solution and that allocates one variable. But also partial instantiations allocating more than one variable can be removed from the search space. A partial instantiation that can not be part of any solution is called a Nogood. If a Nogood allocates more than one variable, it is not implied that each partial instantiation with a subset of these allocations can also not be part of any solution. Nogoods allocating more than one variable can not be removed from the search space by removing values from the domains. For Nogoods allocating one variable, it is possible to remove values from the domains and produce an optimised CSP. These are called unary Nogoods further on. For Nogoods allocating more than one variable, additional information storage is necessary to represent this information. Both forms of Nogoods are utilised in solving methods (Ghedira, 2013, p.29). Domains can be represented as sets of values but also as ranges of values or as sets of ranges of values. They must be finite and must be of discreet mathematical types. The representations with ranges provide a memory reduced representation of domains. When domains are represented as sets of ranges also holes in simple ranges can be represented. This is less memory reduced as with simple ranges representations. For some applications, the simple range representations are already the most practical. Optimising the search space therefor means to tight those ranges. This means to set the lower bound of each range to its lowest possible lower bound and to set the

upper bound of each range to its highest possible upper bound. The generally lowest possible lower bound and the generally highest possible upper bound may be different from those that are meant here. While the generally optimal bounds may already be not contained in the domain to reduce, the optimal bounds in the already limited domain must be determined.

```
1 D = {
2     [1,2],
3     [1,10],
4     [1,4]
5 }
```

LISTING 2.5: Example for textual representation for domains as ranges for the constraint satisfaction problem (CSP) from listing 2.4

```
1 D = {
2     [1,1],
3     [2,3],
4     [3,4]
5 }
```

LISTING 2.6: Optimised ranges for the constraint satisfaction problem (CSP) from listing 2.5

Listing 2.5 shows the corresponding representation of domains as ranges for the CSP from listing 2.4. The optimised ranges are shown in listing 2.6. The upper bound of the second domain, for variable `Pet`, has changed from `10` to `3`. Because the upper bound for variable `Individual` is `4` and the lower bound for variable `Person` is `1`, the first constraint `Individual= Person+ Pet` limits the highest possible upper bound of variable `Pet` to `3`. The upper bound of the first domain, for variable `Person`, has changed from `2` to `1`. Because the highest possible upper bound for variable `Pet` is `3`, the second constraint `Pet≥ 2 * Person` limits the highest possible upper bound of variable `Person` to `1`. The second constraint `Pet≥ 2 * Person` also limits the lowest possible lower bound of the second domain, for variable `Pet`, to `2`, because the lower bound of `Person` is `1`. The lower bound of the third domain, for variable `Individual`, has changed from `1` to `3`. Because the lowest possible lower bounds for variables `Pet` and `Person` are `1` and `2`, the first constraint `Individual= Person+ Pet` limits the lowest possible lower bound of variable `Individual` to `3`.

More efficient algorithms for search space optimisation can be formulated by using only simple ranges for domains instead of explicit sets. The bounds optimisation can be done iteratively until a fix point is reached. This can also result in discovering a contradiction, which means that the CSP does not have any solution. A potential negative aspect is, that holes in the ranges can be not removed from the search space.

Chapter 3

Extension of USE

The background for the productive part of this work was presented up to this point. Some problems with the USE instance finder plugin are already outlined in section 2.5.2. Regarding the potential for UML/OCL model V&V, presented in section 2.5.1, it is obvious that the plugin can be significantly improved by certain extensions. Implemented advanced features for the plugin are presented in the following. The underlying problems and the benefits of the features are also outlined.

One may want to understand the differences or similarities of two or more configurations, when working with multiple instance finder configurations. Currently, this can only be done by manually comparing the configurations. Since the configurations can be constantly changed in the workflow, this must be done repeatedly. But it is not advisable to rely on certain configuration parts not having changed. Configurations should mostly be fully compared. Each individual operation, and therefor the frequent execution of such operations, can be time consuming. For this reason and because the results of manual procedures can be distorted by errors made by the user, an automised solution could greatly increase the usability.

Initial configurations generated by USE contain fixed default values. In most cases such a default configuration does not represent an appropriate search space in terms of bounded model checking. On most of the models, no instances can be found with those default configurations. Additionally the time needed for computing this result is sometimes very short, but also often very long. For this reason, a feature for flexible configuration generation would gain usability.

Configurations can specify inappropriate search spaces in terms of bounded model checking. Simple configuration details may cause the result of not finding any instance. The absence of certain kinds of inconsistencies could be validated automatically. Additionally USE currently allows to specify search space aspects that are obviously contradictory. A feature providing information about the presence of inconsistencies would gain usability. Furthermore, this may be enhanced with providing modification suggestions.

Summarised, three features regarding configuration comparison, clever configuration generation and also the validation of configurations will be presented in the following. Some of the problems are solved using the same concepts. Each of the three subsections deals with one of the features in detail.

3.1 Configuration comparison

Enhanced usability may be achieved with providing computed configuration comparison details. The underlying problem and the benefits of this feature are presented in the following. The following sections contain details and examples of the implemented feature. Configuration comparison is a complex task. The effort depends on the complexity of the model to which the configuration fits. Currently the

differences or similarities between two or more configurations must be worked out manually. The results of this manual procedures can be distorted by errors made by the user. The time needed for this procedure is bound to the number of model elements. The more model elements, the more configuration details exists. With more configuration details, the time needed for this process increases.

After a manual comparison is done, one or more of the configurations may be modified. The comparison may be processed again. The user may mistakenly assume that selected parts of the configurations do not need to be compared again. This may be driven by the need to reduce the amount of time required. Repeated comparison processes factorise the time required, while already the time required for one comparison process is inappropriate high.

The implemented feature provides the automised comparison of selected types of configuration details. Conceptually, different classes of comparison results are defined. Each complete comparison result and also each underlying comparison result of configuration parts is typified. Information about each comparison of each configuration detail is given, but also the classification in the impact on the comparison of the overall configuration. Not all types of configuration details are respected in the comparison. Some are ignored. This is contained in the information, given with the result of a comparison.

The comparison of configurations with the help of the feature leads apparently immediately to a result. A comparison regarding the time required is no longer time-consuming. User errors during manual comparisons are no longer possible.

Additionally it is worth noting that with the information the feature gives, also advanced comparison aspects become easily visible. It is possible to see whether one configuration represents a search space that is a subset of the one represented by another configuration. Overlappings of the search spaces are easy to identify.

The implemented feature is presented in detail in the following subsections. A presentation of the overall concept of the comparison procedure is contained in the first section. The second section presents the partial comparison result types and their merging aspects. Details about how certain configuration parts are compared are contained in section three. The fourth section outlines the additional provided information about the relations of search spaces and the last two sections present the integrations in the user interfaces (UIs).

3.1.1 Stage wise comparison

A special concept is developed for comparing configurations. Always two configurations are compared. Since a configuration has atomary configuration aspects, e.g. the minimum real value, which can also be interpreted combined, e.g. the minimum and maximum real value and its step range, a definition of comparison levels comes in mind. Only on such levels a statement about the comparison can be made. A level summarises the comparison of a set of other levels or atomary configuration aspects. On each level a partial comparison result for the merged partial comparison results of the underlying levels or the results of the comparison of atomary configuration aspects can be evaluated. The top level describes the comparison of all configuration aspects.

Level1 (whole configurations) consists of:

Level1.1 (class specific configurations) , which consists for each class of:

Level1.1.1 (preferred object names) , which compares the set of preferred instance names and

Level 1.1.2 (number of objects for class) , which consists of the minimum and maximum number of objects,

Level1.2 (attribute specific configurations) , which consists for each attribute of:

Level1.2.1 (preferred attribute values) , which compares the set of preferred attribute values,

Level1.2.2 (amount of defined attributes) , which consists of the minimum and maximum number of defined attributes and

Level1.2.3 (amount of collection items) (only for collection type attributes), which consists of the minimum and maximum number of collection items,

Level1.3 (association specific configurations) , which consists for each association of:

Level1.3.1 (required links) , which compares the set of required links and

Level 1.3.2 (number of links) , which consists of the minimum and maximum number of links,

Level1.4 (type specific configurations) , which consists of:

Level1.4.1 (integer type configurations) , which consists of:

Level1.4.1.1 (preferred integer values) , which compares the set of preferred integer values and

Level 1.4.1.2 (range of integer values) , which consists of the minimum and maximum integer values,

Level1.4.2 (string type configurations) , which consists of:

Level1.4.2.1 (preferred string values) , which compares the set of preferred string values and

Level 1.4.2.2 (number of different string values) , which consists of the minimum and maximum number of different string values,

Level1.4.3 (real type configurations) , which consists of:

Level1.4.3.1 (preferred real values) , which compares the set of preferred real values and

Level 1.4.3.2 (range of real values) , which consists of the minimum and maximum real values and the step range,

Level1.5 (invariant specific configurations) , which compares the activation or negation for each invariant and

Level1.6 (option configurations) , which consists of:

Level1.6.1 (`aggregationcyclefreeness`**)** , which compares the `aggregationcyclefreeness` option and

Level1.6.2 (`forbiddensharing`**)** , which compares the `forbiddensharing` option.

On the handling of levels it is differentiated between levels without sub levels and levels with sub levels. For levels with sub levels a general comparison result merging procedure is implemented. This is presented in the next section. For levels without sub levels a specific comparison result computation procedure is implemented. This is also presented in the next but one section.

3.1.2 Classification for partial comparison results

Each of the comparison levels that can be evaluated gives a result on evaluation. Those results are typified. The types represent what a partial comparison result indicates for the comparison result of the parent level and thus for the result of the whole configurations. Comparison is interpreted as an operator with two operands. The ordering of the operands is crucial. Inverting the operands may result in an other type. Each operand represents the configuration aspects of a specific level.

Symbol	Search space relation indicating
`!?`	no
`!!`	no
`..`	no
`!=`	no
`==`	yes
`>=`	yes
`<=`	yes
`lO`	no
`rO`	no
`lD`	no
`rD`	no

TABLE 3.1: Partial comparison result types

Table 3.1 shows the eleven types of comparison results. Only three of the types indicate a search space relation. Two configurations have the same search space if their comparison result is of type ==. When the result is >=, the configuration aspects used as first operand represents a search space that is a super set of the search space represented by the configuration aspects used as second operand. With <= as result type, the configuration aspects used as first operand represents a search space that is a subset of the search space represented by the configuration aspects used as second operand. The symbols must be interpreted as stated in the following:

`!?` declares that the configuration parts compared are not comparable. This can be the case when one configuration contains configuration aspects for model elements which are not contained in the other configuration.

`!!` declares that at least one of the configuration parts compared is not valid in terms of comparison. This can be the case when some configuration aspects are illogical.

`..` declares that the configuration parts are not respected for comparison. With the current implementation, for example the preferred instance names are not respected for comparison.

`!=` declares that the configuration parts compared represent an unequal search space in terms of bounded model checking.

`==` declares that the configuration parts compared represent an equal search space in terms of bounded model checking.

`>=` declares that the configuration parts compared of the first operand represent an search space that is a super set of the search space represented by the configuration parts of the other operand.

`<=` declares that the configuration parts compared of the second operand represent an search space that is a super set of the search space represented by the configuration parts of the other operand.

`lO` declares that the configuration parts compared represent search spaces whose intersection is not empty but which are neither equal nor a subset or super set of each other. This is only applicable for search space domains with elements that are comparable and ordered. It declares that the configuration parts compared of the first operand represent a search space that covers partially the other search space but also preliminary domain elements of the search space represented by the configuration parts of the other operand. This implies that the search space represented by the second operand also contains succeeding domain elements of the other operands search space.

`rO` declares that the configuration parts compared represent search spaces whose intersection is not empty but which are neither equal nor a subset or super set of each other. This is only applicable for search space domains with elements that are comparable and ordered. It declares that the configuration parts compared of the second operand represent a search space that covers partially the other search space but also preliminary domain elements of the search space represented by the configuration parts of the other operand. This implies that the search space represented by the first operand also contains succeeding domain elements of the other operands search space.

`lD` declares that the configuration parts compared represent search spaces whose intersection is empty. This is only applicable for search space domains with elements that are comparable and ordered. It declares that the configuration parts compared of the first operand represent a search space that covers only preliminary domain elements of the search space represented by the configuration parts of the other operand.

`rD` declares that the configuration parts compared represent search spaces whose intersection is empty. This is only applicable for search space domains with elements that are comparable and ordered. It declares that the configuration parts compared of the first operand represent a search space that covers only succeeding domain elements of the search space represented by the configuration parts of the other operand.

	!?	!!	..	!=	==	>=	<=	lO	rO	lD	rD
!?	!?	!?	!?	!?	!?	!?	!?	!?	!?	!?	!?
!!	s.i.	!!	!!	!!	!!	!!	!!	!!	!!	!!	!!
..	s.i.	s.i.	..	!=	==	>=	<=	lO	rO	lD	rD
!=	s.i.	s.i.	s.i.	!=	!=	!=	!=	!=	!=	!=	!=
==	s.i.	s.i.	s.i.	s.i.	==	>=	<=	!=	!=	!=	!=
>=	s.i.	s.i.	s.i.	s.i.	s.i.	>=	!=	!=	!=	!=	!=
<=	s.i.	s.i.	s.i.	s.i.	s.i.	s.i.	<=	!=	!=	!=	!=
lO	s.i.	s.i.	s.i.	s.i.	s.i.	s.i.	s.i.	lO	!=	!=	!=
rO	s.i.	s.i.	s.i.	s.i.	s.i.	s.i.	s.i.	s.i.	rO	!=	!=
lD	s.i.	s.i.	s.i.	s.i.	s.i.	s.i.	s.i.	s.i.	s.i.	lD	!=
rD	s.i.	s.i.	s.i.	s.i.	s.i.	s.i.	s.i.	s.i.	s.i.	s.i.	rD

"s.i." as "see inverted" means to look up the specification with inverted roles.

TABLE 3.2: Partial comparison result types for merged partial comparison results

Table 3.2 shows the comparison result types resulting in merging two partial results of specific comparison result types. There it is also interpreted as an operator with two operands. Each operand is a comparison result type. The upper row and the left column contain all result types each. The cells under the upper row and right of the left row contain the comparison result type for merging the type used as first operand, indicated by the cell in the left column and the same row, with the type used as second operand, indicated by the cell in the upper row and the same column. The matrix shown in table 3.2 is inverted. This means applying the merge operator with inverted operands results in the same result type.

Only the merge results for merging two comparison result types are provided with table 3.2. But there are levels that need to merge one or more than two result types. So the general merge operator must have n operands with $n \geq 1$. With only one operand, its merge result type is the one given by the operand. For two operands, the result type is specified in table 3.2. Merging more than two result types can be done by merging two of the result types and then merge this result again with another result type and so on. The order of selection of to be merged result types is arbitrary.

Since it is not obvious that the order of selection of to be merge result types is arbitrary, this must be proofed formally. This is done in appendix B.

3.1.3 Configuration attribute specific comparisons

Here, the stage wise comparison concept and the concept of classification for partial comparison results is complemented by the level specific partial comparison procedures. Those are presented in the following.

There are two types of levels. Firstly, there are levels that only have sub levels. Secondly, there are levels for comparison of specific configurations parts. For levels with sub levels, the procedure is already presented with the partial comparison result merging. For the levels of the other type, there are specific comparison processes. These are presented here.

Level	Possible partial comparison result types
Level 1.1.1	{..}
Level 1.2.1	{..}
Level 1.2.2	{..}
Level 1.2.3	{..}
Level 1.3.1	{..}
Level 1.4.1.1	{..}
Level 1.4.2.1	{..}
Level 1.4.3.1	{..}
Level 1.5	{..}
Level 1.6.1	{..}
Level 1.6.2	{..}

TABLE 3.3: Possible partial comparison result types of levels that are not respected with the implemented configuration comparison procedure

Level	Possible partial comparison result types
Level 1.1.2	{!!, ==, >=, <=, lD, rD, lO, rO, !=}
Level 1.3.2	{!!, ==, >=, <=, lD, rD, lO, rO, !=}
Level 1.4.1.2	{!!, ==, >=, <=, lD, rD, lO, rO, !=}
Level 1.4.2.2	{!!, ==, >=, <=, lD, rD, lO, rO, !=}
Level 1.4.3.2	{!!, ==, >=, <=, lD, rD, lO, rO, !=}

TABLE 3.4: Possible partial comparison result types of levels that are respected with the implemented configuration comparison procedure

Some levels are not respected for the implemented configuration comparison procedure. They always have .. as partial comparison result. Table 3.3 shows all such levels. For the other levels, shown in table 3.4, the specific comparison processes are presented in the following:

Level 1.1.2 (number of objects for class) has result

- `!!` when one of the minimum or maximum object numbers is less than 0 or the minimum is greater than the maximum for a configuration.
- `==` when not `!!` and both minimum and minimum as well as maximum and maximum object numbers are equal.
- `>=` when not `!!` and minimum object number of the first configuration is less than minimum object number of the second configuration and maximum object number of the first configuration is greater than maximum object number of the second configuration.

`<=` when not `!!` and minimum object number of the first configuration is greater than minimum object number of the second configuration and maximum object number of the first configuration is less than maximum object number of the second configuration.

`lD` when not `!!` and maximum object number of the first configuration is less than minimum object number of the second configuration.

`rD` when not `!!` and maximum object number of the second configuration is less than minimum object number of the first configuration.

`lO` when not `!!` and minimum object number of the first configuration is less than minimum object number of the second configuration and maximum object number of the first configuration is less than maximum object number of the second configuration and is also greater than or equal the minimum object number of the second configuration.

`rO` when not `!!` and minimum object number of the second configuration is less than minimum object number of the first configuration and maximum object number of the second configuration is less than maximum object number of the first configuration and is also greater than or equal the minimum object number of the second configuration.

`!=` elsewise.

Level 1.3.2 (number of links) has result

`!!` when one of the minimum link numbers is less than 0 or one of maximum links specifications is less than -1 or the minimum is greater than the maximum for a configuration when maximum is not -1.

`==` when not `!!` and both minimum and minimum as well as maximum and maximum link numbers are equal.

`>=` when not `!!` and minimum link number of the first configuration is less than minimum link number of the second configuration and maximum link number of the first configuration is -1 or greater than maximum link number of the second configuration and maximum link number of the second configuration is not -1.

`<=` when not `!!` and minimum link number of the second configuration is less than minimum link number of the first configuration and maximum link number of the second configuration is -1 or greater than maximum link number of the first configuration and maximum link number of the first configuration is not -1.

`lD` when not `!!` and maximum link number of the first configuration is not -1 and less than minimum link number of the second configuration.

`rD` when not `!!` and maximum link number of the second configuration is not -1 and less than minimum link number of the first configuration.

`lO` when not `!!` and minimum link number of the first configuration is less than minimum link number of the second configuration and maximum link number of the first configuration is not -1 and less than maximum link number of the second configuration if that is not -1 and is also greater than or equal the minimum link number of the second configuration.

`rO` when not `!!` and minimum link number of the second configuration is less than minimum link number of the first configuration and maximum link number of the second configuration is not -1 and less than maximum link number of the first configuration if that is not -1 and is also greater than or equal the minimum link number of the first configuration.

`!=` elsewise.

Level 1.4.1.2 (range of integer values) has result

`!!` when only one of the integer number configurations is enabled or the minimum integer number is greater than the maximum integer number for a configuration.

`==` when not `!!` and both of the integer number configurations are enabled and both minimum and minimum as well as maximum and maximum integer numbers are equal.

`>=` when not `!!` and minimum integer number of the first configuration is less than minimum integer number of the second configuration and maximum integer number of the first configuration is greater than maximum integer number of the second configuration.

`<=` when not `!!` and minimum integer number of the first configuration is greater than minimum integer number of the second configuration and maximum integer number of the first configuration is less than maximum integer number of the second configuration.

`lD` when not `!!` and maximum integer number of the first configuration is less than minimum integer number of the second configuration.

`rD` when not `!!` and maximum integer number of the second configuration is less than minimum integer number of the first configuration.

`lO` when not `!!` and minimum integer number of the first configuration is less than minimum integer number of the second configuration and maximum integer number of the first configuration is less than maximum integer number of the second configuration and is also greater than or equal the minimum integer number of the second configuration.

`rO` when not `!!` and minimum integer number of the second configuration is less than minimum integer number of the first configuration and maximum integer number of the second configuration is less than maximum integer number of the first configuration and is also greater than or equal the minimum integer number of the second configuration.

`!=` elsewise.

Level 1.4.2.2 (number of different string values) has result

`!!` when only one of the different string values configurations is enabled or one of the minimum or maximum number of different string values is less than 0 or the minimum is greater than the maximum for a configuration.

`==` when not `!!` and both of the different string values configurations are enabled and both minimum and minimum as well as maximum and maximum number of different string values are equal.

`>=` when not `!!` and minimum number of different string values of the first configuration is less than minimum number of different string values of

the second configuration and maximum number of different string values of the first configuration is greater than maximum number of different string values of the second configuration.

`<=` when not `!!` and minimum number of different string values of the first configuration is greater than minimum number of different string values of the second configuration and maximum number of different string values of the first configuration is less than maximum number of different string values of the second configuration.

`lD` when not `!!` and maximum number of different string values of the first configuration is less than minimum number of different string values of the second configuration.

`rD` when not `!!` and maximum number of different string values of the second configuration is less than minimum number of different string values of the first configuration.

`lO` when not `!!` and minimum number of different string values of the first configuration is less than minimum number of different string values of the second configuration and maximum number of different string values of the first configuration is less than maximum number of different string values of the second configuration and is also greater than or equal the minimum number of different string values of the second configuration.

`rO` when not `!!` and minimum number of different string values of the second configuration is less than minimum number of different string values of the first configuration and maximum number of different string values of the second configuration is less than maximum number of different string values of the first configuration and is also greater than or equal the minimum number of different string values of the second configuration.

`!=` elsewise.

Level 1.4.3.2 (range of real values) has result

`!!` when only one of the real number configurations is enabled or the minimum real number is greater than the maximum real number for a configuration or the one of the step values is less than or equal 0.

`!=` when the step values are not equal or the comparison has no other result.

`==` when not `!!` and step values are equal and both of the real number configurations are enabled and both minimum and minimum as well as maximum and maximum number of real number.

`>=` when not `!!` and step values are equal and minimum real number of the first configuration is less than minimum real number of the second configuration and maximum real number of the first configuration is greater than maximum real number of the second configuration.

`<=` when not `!!` and not step values are equal and minimum real number of the first configuration is greater than minimum real number of the second configuration and maximum real number of the first configuration is less than maximum real number of the second configuration.

`lD` when not `!!` and not step values are equal and maximum real number of the first configuration is less than minimum real number of the second configuration.

`rD` when not `!!` and not step values are equal and maximum real number of the second configuration is less than minimum real number of the first configuration.

`lO` when not `!!` and not step values are equal and minimum real number of the first configuration is less than minimum real number of the second configuration and maximum real number of the first configuration is less than maximum real number of the second configuration and is also greater than or equal the minimum real number of the second configuration.

`rO` when not `!!` and not step values are equal and minimum real number of the first configuration is less than minimum real number of the second configuration and maximum real number of the first configuration is less than maximum real number of the second configuration and is also greater than or equal the minimum real number of the second configuration.

3.1.4 Relationship analysis

The search space represented by the USE instance finder configurations has a lot of dimensions. Visualisation of all dimensions at once is very complex. There may also be some possible inferred knowledge as object of interest. Special perspectives on configurations may meet the requirements of the user. One of these perspectives focuses on the relations between the search spaces represented by configurations. In the common workflow with the USE instance finder, one may work for a model with several configurations simultaneously. Not only the relationships between two configurations but also between more than two configurations are object of interest.

It is already provided that the relationships regarding the search space are computed based on the comparison result types. This allows to identify families of search spaces represented by the configurations. Each two family members are either equal (==) or one is the subset (<=) or super set (>=) of the other.

The implemented features provide a UI specific informative output, informing about the computed families. This can be used to focus on the comparison details of only family members in another comparison process. The displayed information is then reduced and should make it easier to understand.

3.1.5 Command-line interface integration

The presented comparison concept and the procedure are implemented in USE. It extends the functionality of the USE Shell, the CLI. This is explained in the following and then complemented by an example.

Prerequisites for using the functionality are an UML/OCL model loaded in USE and an existing model validator configuration file for the model. The comparison command is available as `modelvalidator -compareConfig` and `mv -cc`. As mandatory argument the path to the configuration file must be specified. Optionally the respected configurations from the file can be selected by specifying the names as arguments. When no (valid) names are specified, all configurations from the file are compared. Specified names are validated before comparison. When a name is specified twice, it will be only used once in the comparison process. If a name is specified that is not contained in the file, it is ignored. In both cases an informative output is given. When (valid) names are specified, only the corresponding configurations

from the file are compared. Of the set of to be compared configurations, each configuration will be compared with each other configuration but not with itself. No comparison is done with inverted operands.

The result of the comparison procedure of two configurations is expressed textually. The text contains a headline with the two configuration names and the symbol for their comparison result. For each result type there is a section. It lists results of levels of comparison comparing actual configuration aspects. Those levels are listed in table 3.3 and table 3.4. Empty sections are not displayed. Displayed sections are always ordered as in table 3.1. For each level specific partial comparison result, listed under a section, additional information are given. The configuration attributes, token into account for partial comparison, are listed in a sentence that explains the partial result. Additionally, the actual values of the configuration attributes are listed pairwise for both configurations in brackets on the end of the sentence.

The result of the comparison procedure of multiple configurations pairwise is also expressed textually. For each pair of configurations, the representation is as explained before but numbered. Each headline has the number as prefix. Then a table for an overview is given. The table contains the numbers of the comparisons, the result symbol and the names of the two configurations compared. Because there may be more text produced than it can be displayed on common screen sizes, this is placed at the end of the output. This gives the most general information. After that, the families are listed as lists of configuration names. Then the user can find the parts of interest manually by scrolling through the details.

The textual representations of all possible values that can be displayed in the textual comparison output do not need much space. For values of collection types, which are only given for attribute that are not respected in the implementation, the textual representation can become extremely large. This is the case, e.g., when listing all elements of a collection. For this reason, the current concept of textual representation may not be an adaptable on implementing comparison procedures for collection type configuration attributes.

```
1  [config1]
2
3  Integer_min = -10
4  Integer_max = 10
5
6
7  [config2]
8
9  Integer_min = -100
10 Integer_max = 100
11
12
13 [config3]
14
15 Integer_min = 0
16 Integer_max = 10
17
18
19 [config4]
20
21 Integer_min = 0
22 Integer_max = 100
23
24
25 [config5]
26
27 Integer_min = 0
```

```
28 Integer_max = 1000
29
30
31 [config6]
32
33 Integer_min = -10
34 Integer_max = 0
35
36
37 [config7]
38
39 Integer_min = 100
40 Integer_max = 1000
41
42 [config8]
43
44 Integer_min = -100000
45 Integer_max = -10000
```

LISTING 3.1: UML-based Specification Environment textual instance finder configurations for model from fig. 2.1 used for comparison

Listing 3.1 shows the content of a configuration file. The contained configurations are used for configuration. Listing A.1 (s. appendix A) shows the results. The configuration is reduced to representative aspects. Only the minimum and maximum integer value is specified in each configuration. In the following some noteworthy comparison details are outlined.

`config1 <= config2` because all other aspects are ignored or equal but the integer configuration aspects indicate that `config2` includes `config1`.

`config2 != config5` because the integer configuration aspects comparison results in `lO` but there are equal aspects.

`config5 != config6` because the integer configuration aspects comparison results in `rO` but there are equal aspects.

`config6 != config7` because the integer configuration aspects comparison results in `lD` but there are equal aspects.

One family, not containing `config8`**,** is found. Each of the configurations is related to at least one of the other family members. `config8` is `!=` with all other configurations.

3.1.6 Graphical user interface integration

The comparison functionality is also integrated in the GUI. This is explained in the following and then complemented by adapting the example from the previous section.

Prerequisites for using the functionality are an UML/OCL model loaded in USE and an existing model validator configuration for the model loaded with the model validator plugin. The comparison command is available as menu item `Compare`, in menu `Configuration`, in the menu bar at the top of the model validator window. A dialog to specify the configuration name opens on selection of the menu item. If successfully a name is specified the comparison window opens. The model validator window will freeze until the comparison window is closed. Initially all configurations from the state in the model validator are compared. While the configurations

may have been loaded from a file, it can contain unsaved changes. The state with changes is used for comparison.

The comparison window has a configuration selection area on the left, with function buttons on the bottom. The comparison overview is the main part of the window. Initially all configurations are selected and the overview contains the comparison overview for comparing all configurations. An overview is given with a matrix. The upper row and left column each contain all selected configurations names. Cells under the upper row and right of the left column contain each the comparison result symbol for the configurations indicated by the left cell in the row and the upper cell in the column. Inverted comparison and comparison with configurations itself is processed.

In the selection area, a selection of configurations to be compared can be made. Clicking the button is necessary for applying the selection to update the overview. Single selection can be made by clicking on the listed name. Multi line selection can be made holding the shift button on keyboard and then clicking on the listed name. All names are selected, starting with the lastly selected name and ending with the now selected name. The present selection can be expanded via multi selection by holding the command button on keyboard and then clicking on the listed name. Additional selection options are accessible via the corresponding button. Options to select all names or deselect all names are provided. Here, also the found families, regarding the compared section currently displayed in the overview area, are listed. For each family, there is an option to only select all family members.

Comparison details are here also given textually in the form that is described in the previous section. Each comparison details of the pairwise comparison are accessible via double click in the corresponding cells in the matrix in the overview area. Details will be provided in a separate comparison details window. The comparison overview window will freeze until the comparison details window is closed. The comparison details window contains the textual representation of the comparison details.

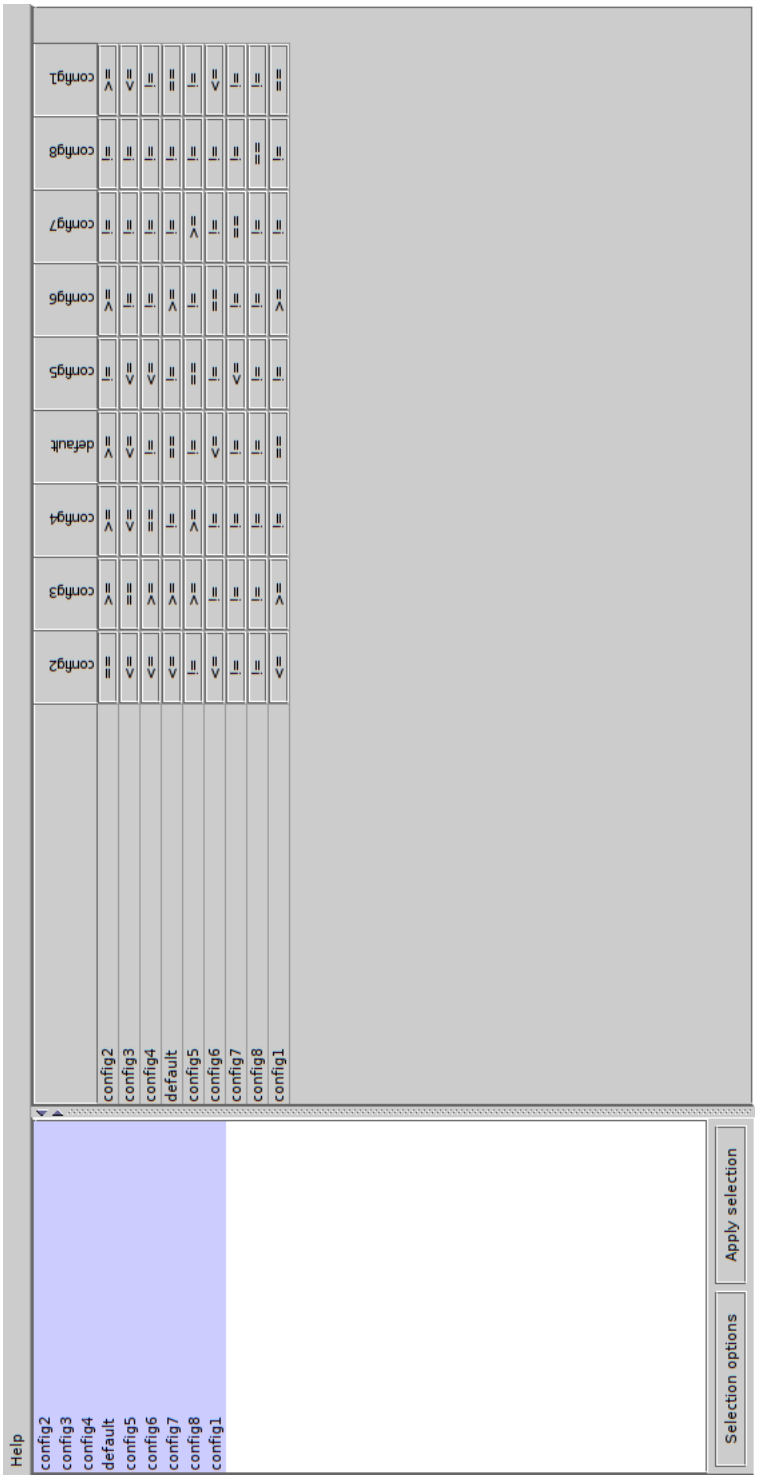

FIGURE 3.1: UML-based Specification Environment initial comparison overview for comparing configurations from listing 3.1

FIGURE 3.2: UML-based Specification Environment comparison overview selection options for example shown in fig. 3.1

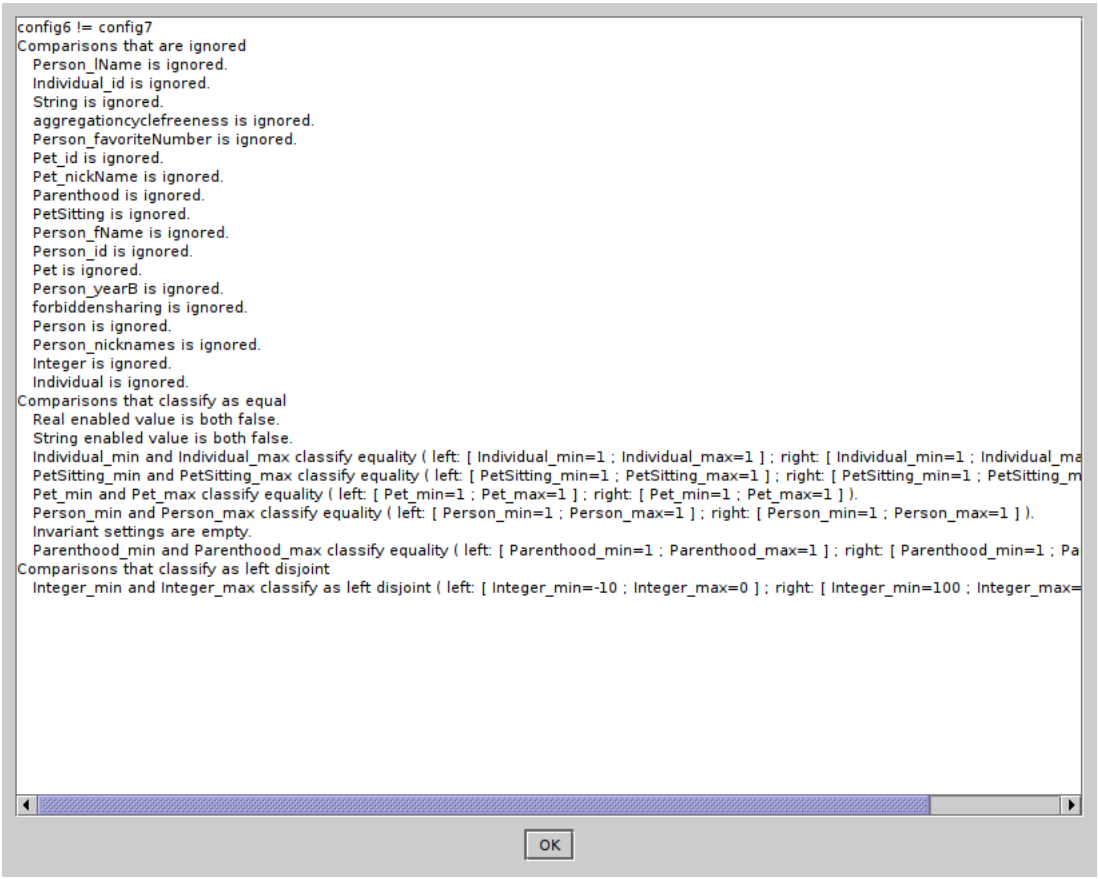

FIGURE 3.3: UML-based Specification Environment initial comparison overview for comparing configurations from listing 3.1

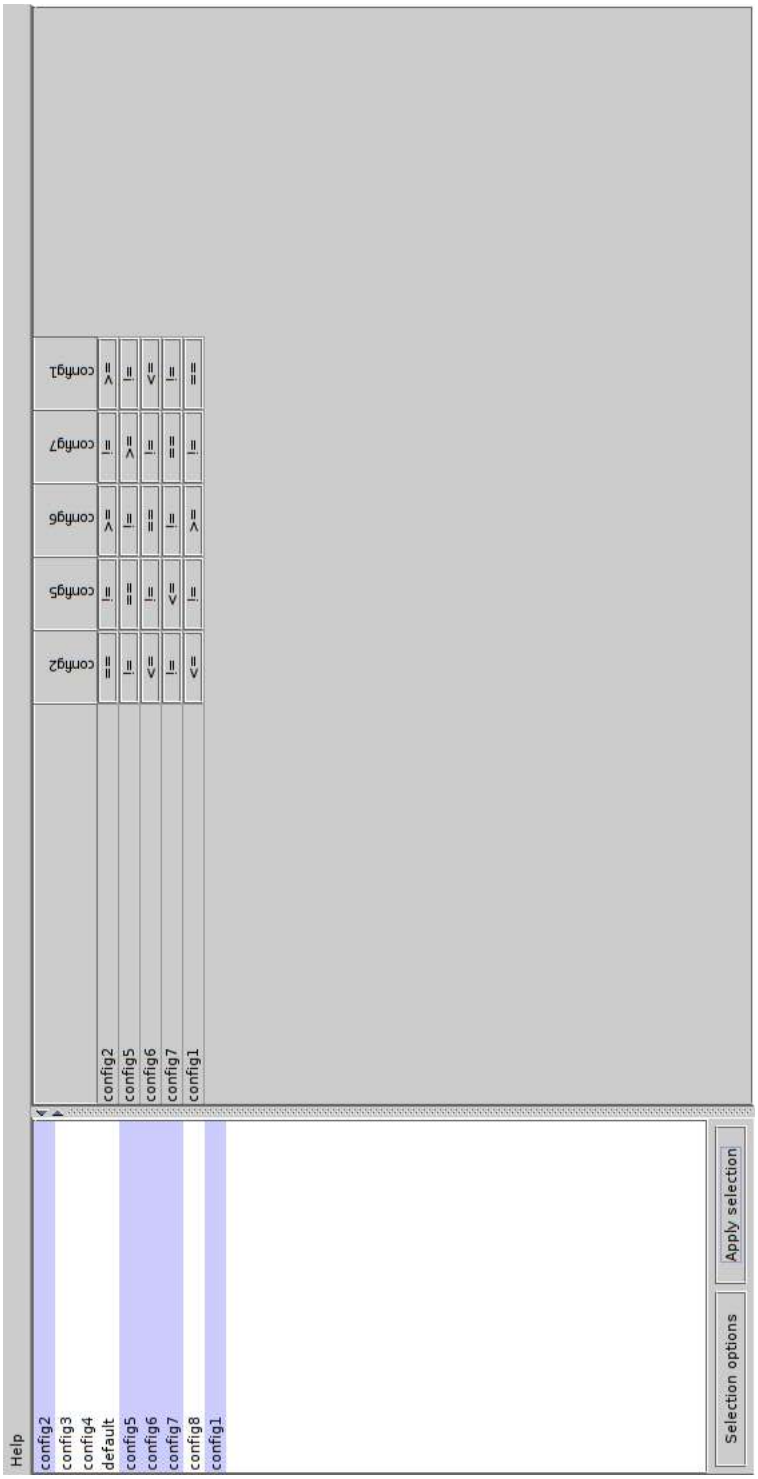

FIGURE 3.4: UML-based Specification Environment comparison overview for comparing selected configurations from listing 3.1

Figure 3.1 shows the initial comparison window for comparing configurations from listing 3.1. All configurations are selected initially and therefor the overview shows the comparisons of all contained configurations. Figure 3.2 shows the selection options given for the comparison window content. The option to select the found family is hovered. The comparison details window opens when double clicking on the corresponding comparison result type symbol in a cell of the matrix in the

overview area. It is shown in fig. 3.3. The overview panel potentially exceeds a fully displayable size. If the window is too small, the overview area becomes scrollable in vertical and horizontal dimensions. With selecting less configurations to be compared, the information to display in the overview area decreases and therefor also the size needed decreases. Figure 3.4 shows the comparison window in which only a subset of the configurations is selected.

3.2 Constraint satisfaction problems applied for generation of configurations

USE already provides functionality to generate configurations with fixed default values. The problems with this and the potential benefits of a more complex generation feature are covered here. A clever generation feature is implemented. The underlying problem and the benefits of this feature are presented in the following. The following sections contain details and examples of the implemented feature.

Initial configurations, generated by USE, contain fixed default values. In most cases, such a default configuration does not represent an appropriate search space in terms of bounded model checking. On most of the models, no instances can be found with those default configurations. Additionally, the time needed for computing this result is sometimes very short, but also often very long.

The implemented feature provides the definition of element specific initial bounds. This can be done on different levels. Initial bounds can be specified generally for all model elements. They also can be specified generally for all model elements but specifically for classes or one class and so on. The precision of the initial bounds is selected by the user. The specification solely on the top level provides a simple and fast option. With the initial bounds a bounds tightening procedure is started. The result is a configuration that has exactly the initial bounds or more tight bounds. The bounds tightening process reduces the search space for model instance finding. It does not reduce the set of model instances to be found. Since the implemented bounds tightening process can only be applied for finite domains, the initial bounds specification can not contain unlimited domains. This would be possible in the configurations. The implemented feature provides options to overwrite attribute bounds with special values, representing boundlessness, after the bounds tightening procedure.

There may be other and simpler possibilities to solve the underlying problem. The applied concepts are adopted for other features. Therefor, the costs for implementation are shared. The implemented feature is very complex but also provides various precision possibilities with the flexible initial bounds specification.

There are several benefits of the implemented feature. First of all, the bounds tightening may not be possible due to inconsistencies in the model or inappropriate initial bounds. To verify that it is not caused by inappropriate initial bounds, the clever generation process can be started again with most general initial bounds. If bounds tightening is not possible once again, obviously the model contains inconsistencies so that model instances can not exist. Another benefit is that the bounds tightening eliminates search space. Therefor, the model instance finding process may become faster. The implemented feature also enables all type settings for types that are used in the model. The present default generation process does not enable model specific types. Therefor, this must be enabled by hand. The implemented feature disables also all type settings for types that are not used in the model.

The implemented feature is presented in detail in the following subsections. The first section presents the crucial concept of bounds tightening applied for the generation procedure. Adjustment options, that are applied after bounds tightening, are presented in the second section. The last two sections present the integrations in the UIs.

3.2.1 Bounds tightening for configurations with derived model specific attributes

Bounds tightening is achieved using CSPs with simple ranges for variable domains. The concept of bounds tightening is presented in the following. Variables for all model elements are taken, representing the minimum and maximum number of instances. Constraints are derived from several other model aspects, e.g. the relations of classes and the multiplicities of associations. On the resulting CSP, a redundant search space removal procedure is applied. The variable domains in the resulting optimised CSP are then transferred to configuration aspects for the corresponding model elements. Theses steps are presented in detail in the following.

The specification of variables and their domains is the first step of the bounds tightening procedure. A variable is created for each class and association. A variable is also created for each attribute that is owned by a class. This differs from the concept in the configurations. There are settings for a variable only on most general level in the configuration. Here, also for abstract classes variables are created. Regarding classes and associations the variables represent the number of instances of these elements. The variables for attributes represent the numbers of objects with value for the attribute. Only finite domains for variables are allowed. The domains are represented as simple ranges. The value -1 for attribute lower bounds and attribute and association upper bounds can not be expressed. Configuration aspects for type domains, invariants and options are not included in the bounds tightening process. The initial domains for all variables here must be specified using the corresponding UI.

Adding model specific constraints to the CSP is the second step of the bounds tightening procedure. There a four kinds of constraints. There are constraints for generalisations. Secondly, there are constraints for the dependencies of the number of defined attributes to the amount of owning classes instances. Thirdly, there are constraints for the dependencies of the number of links regarding the multiplicities and for the numbers of associated objects. Lastly, there are constraints derived from the model invariants. A constraint is added for each class with subclasses. The number of objects for a parent class must be equal to the sum of numbers of objects of its child classes. At least one constraint is added for each attribute. The number of objects of each attributes owning class must be greater than or equal the number of defined attributes. For each binary association up to four constraints are added, because for each of the two association ends up to two constraints are added. The number of links must be greater than or equal the minimum multiplicity of the association end times the number of objects of the class of the other association end. If the multiplicity is unbound, 0 is used as minimum multiplicity. The number of links must be less than or equal the maximum multiplicity of the association end times the number of objects of the class of the other association end. If the multiplicity is unbound, no constraint for the maximum multiplicity is added. If the minimum and maximum multiplicity is equal, only one corresponding constraint is added. Then the number of links must be equal to the equal minimum and maximum multiplicity of the association end times the number of objects of the class of the other association

end. Associations with more than two ends are currently ignored. No constraints are added for model invariants. Deriving constraints from model invariants is very complex, because there are various special cases to handle. Clarisó, González, and Cabot (2019) already presented a method to generate OCL expression specific CSP constraints for invariants. The current implementation may be extended to include this method.

An external Java framework for CSP functionality is used for the domain optimisation. The Choco Solver version 4.10.2 is used. It is an open-source Java library for constraint programming. The generated CSP is translated to a structure defined in the framework. The optimisation is processed by using the appropriate functionality provided by the framework.

Adopting the variables domains of the optimised CSP in a configuration is the last step. Firstly, a configuration is generated with the default generation process. This configuration is enriched by the bounds represented by the model elements variables domains. For classes and associations the minimum and maximum numbers of instances are set to the domain bounds of the corresponding CSP variables. The inherited attributes of all classes are not respected in this process. Only the attributes directly defined are respected for each class. The minimum and maximum numbers of defined attributes are set to the domain bounds of the corresponding CSP variables. As last action, all needed type settings are enabled. Type settings gets enabled when there exists an attribute of that type, whose maximum defined value is not 0. Otherwise they get disabled. Several configuration aspects are unchanged. In detail, all values for preferred and required values settings, the values for integer and real minimum and maximum settings, the value for real step setting, the values for string minimum and maximum number of different values settings and the option settings and invariant settings, as well as the values for minimum and maximum number of elements for collection type attributes are unchanged.

At this point, a configuration is already generated. The value -1 for attribute lower bounds and attribute and association upper bounds is not supported until the generation procedure reaches this point. The clever generation procedure supports additional options for actions applied after the CSP bounds tightening result is transferred to a configuration. The first option is to set all attribute lower bounds to -1. The second option is to set all attribute upper bounds to -1 when they are greater than or equal the upper bound of their owning class. An attribute specific option choice is as complex as processing those attribute specific modifications manually. This is why its scope has been kept so general.

```
1  V = {
2     Class.Person.Attribute.favoriteNumber,
3     Class.Person.Attribute.id,
4     Class.Individual.Attribute.id,
5     Class.Pet.Attribute.id,
6     Class.Individual,
7     Class.Person.Attribute.lName,
8     Class.Person.Attribute.yearB,
9     Class.Pet.Attribute.nickName,
10    Class.Person.Attribute.nicknames,
11    Class.Person.Attribute.fName,
12    Association.PetSitting,
13    Association.Parenthood,
14    Class.Person,
15    Class.Pet
16 }
17
18 D = {
```

```
19        [1,100],
20        [1,100],
21        [1,100],
22        [1,100],
23        [1,100],
24        [1,100],
25        [1,100],
26        [1,100],
27        [1,100],
28        [1,100],
29        [1,100],
30        [1,100],
31        [1,100],
32        [1,100]
33    }
34
35    C = {
36    "Association.PetSitting = 1 * Class.Pet"
37    "Class.Person >= Class.Person.Attribute.yearB"
38    "Association.Parenthood >= 0 * Class.Person"
39    "Association.Parenthood >= 0 * Class.Person"
40    "Association.PetSitting >= 2 * Class.Person"
41    "Class.Person >= Class.Person.Attribute.lName"
42    "Class.Person >= Class.Person.Attribute.favoriteNumber"
43    "Class.Person >= Class.Person.Attribute.id"
44    "Association.Parenthood <= 2 * Class.Person"
45    "Class.Pet >= Class.Pet.Attribute.nickName"
46    "Class.Individual = Class.Pet + Class.Person"
47    "Class.Person >= Class.Person.Attribute.nicknames"
48    "Class.Person >= Class.Person.Attribute.fName"
49    "Class.Pet >= Class.Pet.Attribute.id"
50    "Class.Individual >= Class.Individual.Attribute.id"
51    }
```

LISTING 3.2: Constraint satisfaction problem for model from fig. 2.1 with general one to 100 domains

```
18    D = {
19        [1,49],
20        [1,49],
21        [1,100],
22        [1,99],
23        [3,100],
24        [1,49],
25        [1,49],
26        [1,99],
27        [1,49],
28        [1,49],
29        [2,99],
30        [1,98],
31        [1,49],
32        [2,99]
33    }
```

LISTING 3.3: Optimised domains of constraint satisfaction problem for model from fig. 2.1 with general one to 100 domains

```
1   [config1]
2
3   Integer_min = -10
4   Integer_max = 10
5
6   String_max = 10
```

```
7
8  Real_min = -2.0
9  Real_max = 2.0
10 Real_step = 0.5
11
12 # ------------------------------------------------------------------------
       Individual
13
14 Individual_id_min = -1
15 Individual_id_max = -1
16
17 # ------------------------------------------------------------------------
       Person
18 Person_min = 1
19 Person_max = 49
20
21 Person_fName_min = -1
22 Person_fName_max = -1
23 Person_favoriteNumber_min = -1
24 Person_favoriteNumber_max = -1
25 Person_lName_min = -1
26 Person_lName_max = -1
27 Person_nicknames_min = -1
28 Person_nicknames_max = -1
29 Person_nicknames_minSize = 0
30 Person_nicknames_maxSize = -1
31 Person_yearB_min = -1
32 Person_yearB_max = -1
33
34 # Parenthood (parent:Person, child:Person) - - - - - - - - - - - - - - - -
       - - - - -
35 Parenthood_min = 1
36 Parenthood_max = 98
37
38 # PetSitting (sitter:Person, pet:Pet) - - - - - - - - - - - - - - - - - -
       - - - - -
39 PetSitting_min = 2
40 PetSitting_max = 99
41
42 # ------------------------------------------------------------------------
       Pet
43 Pet_min = 2
44 Pet_max = 99
45
46 Pet_nickName_min = -1
47 Pet_nickName_max = -1
48 # ------------------------------------------------------------------------

49 aggregationcyclefreeness = on
50 forbiddensharing = on
```

LISTING 3.4: Optimised UML-based Specification Environment instance finder configuration for model from fig. 2.1 with general one to 100 domains

Using as most general domain specification a range from one to 100, the CSP shown in listing 3.2 is generated for the previously presented UML class diagram

model. Constraints for the generalisation of `Pet` and `Person` with `Individual`, constraints for all attributes of all classes and constraints for all association ends are contained in this example. Beside the simple constraints for generalisation and dependencies of attributes to classes, the constraints for association ends are noteworthy. The end `child` of association `Parenthood` with multiplicity at least zero and the end `parent` of association `Parenthood` with multiplicity zero up to two result in the two constraints constraining the number of links to be greater than or equal zero. The end `parent` also constraining the number of links to be less than or equal two times the number of objects of class `Person`. The end `sitter` of association `PetSitting` with multiplicity one constrains the number of links to exactly the number of objects of class `Pet`. The other end `pet` with multiplicity two up to four or six or more than six does not sole constrain the maximum number of links. The minimum number of links is constrained to be at least two times the number of objects for class `Person`.

Applying the bounds tightening procedure, the domains are reduced. Listing 3.3 shows the optimised CSP domains for listing 3.2. Since there is at least one object for `Person`, there must be two links of `PetSitting` and therefor two objects of class `Pet`. In addition, it follows that there must be at least three objects of class `Individual`. Since there is at least one object of class `Person`, there can only be 99 objects of class `Pet`, because they are all objects of class `Individual` and there can only be 100 objects of class `Individual`. Since there can only be 99 objects of class `Pet`, there also can only be 99 links of `PetSitting` and therefor there can only be 49 objects of class `Person`.

The numbers of objects of class `Pet` is at least two times the number of objects of class `Person` and therefor there can exist only up to 33 objects of class `Person` when there can only exits 100 objects of class `Individual`. This is not covered with the given constraints. Advanced constraint generation or further processing of the constraints may result in even tighter domains. The space of different combinations of domain values represented by the CSP variables from listing 3.2 is in listing 3.3 nevertheless already reduced from $100^{14} = 10^{28}$ to $100 * 99^2 * 98^4 * 49^7 = 6,13122497554589853891984 * 10^{25}$. This means an reduction by at least factor 163.

A configuration adopting the optimised domains is shown in listing 3.4. The options to conditionally set attribute lower and upper bounds to -1 are applied. The minimum and maximum numbers of links are adopted from the CSP domains for both associations. For the two non abstract classes the configurations aspects are adopted but not for the abstract class. The computed information for abstract classes is discarded with the current implementation of the model validator plugin. Since all three types are used for attributes of classes which may have instances, there are configuration aspects for all three types.

3.2.2 Command-line interface integration

The presented configuration optimisation and generation concepts are implemented in USE. It extends the functionality of the USE Shell, the CLI. This is explained in the following.

Prerequisite for using the functionality is an UML/OCL model loaded in USE. The clever generation command is available as `modelvalidator -createSmartConfig` and `mv -csc`. As mandatory argument, the path to a configuration file must be specified. An already existing file at that path will not be read but may be overwritten. Optionally flags can be specified. The invariants are respected on generating the CSP constraints with `-respectOclConstraints` and `-roc`. This is not supported yet.

Specifying the flag will currently result in a corresponding information and premature discontinuance. With `-mandatorizeAttributes` and `-ma` the generated configuration will have value -1 for all minimum numbers of defined attributes. With `-generalizeAttributeUpperBounds` and `-gab` the generated configuration will have value -1 for all maximum numbers of defined attributes, if the computed number is equal or greater than the maximum number of instances of the owning class. Finally the initial bounds must be specified. Two values are mandatory. The first value is the general lower bound and the second value is the general upper bound. More precise initial bounds can be optionally specified afterwards. For all model elements and the three types of model elements an option is provided. More precise bounds can be specified with `<option>.[<lb>,<ub>]`. `<lb>` and `<ub>` must be replaced by the lower and upper bound values. The value of the upper bound must be equal or greater than the value of the lower bound. `<option>` must be replaced by a model elements option. `Classes`, `Attributes` and `Associations` are the options to specify general bounds for kinds of model elements. To be more precise, the options `Class.<class>` for classes, `Class.<class>.Attribute.<attribute>` for attributes and `Association.<association>` for associations are provided. `<class>`, `<attribute>` and `<association>` must be replaced by the model elements name. The bounds tightening procedure may fail because there can be specified initial bounds for that some CSP constraints are contradictory. In this case, no configuration is generated but the appropriate information is given. Elsewise, a configuration is generated and saved as a file at the specified path.

3.2.3 Graphical user interface integration

The clever generation functionality is also integrated in the GUI. This is explained in the following and then complemented by adapting the example from the previous section.

Prerequisite for using the functionality is an UML/OCL model loaded in USE. The clever generation command is available as menu item `New (clever)` in menu `Configuration`, in the menu bar at the top of the model validator window. A dialog to specify the configuration name opens on selection of the menu item. The clever generation window opens if successfully a name is specified. The model validator window will freeze until the clever generation window is closed. The clever generation window contains an option area at the top. For each of the three options that are also supported on CLI, there is a checkbox. Initially only the checkbox for enabling respecting OCL invariants for the CSP constraints is disabled. When enabling the option, currently an error dialog pops up, which informs about that this functionality is not supported. The option disables itself. The main area displays the initial bounds specification. For all model elements and the three model elements types, there is firstly a checkbox for enabling precise specification or for disabling. Disabling the checkbox means to adopt the more general specification. For all these and for the most general initial bounds specification, there are two number input fields. Initially all deactivatable specifications are deactivated and the most initial bounds are set to the range from one to 100. On the bottom of the window there is a button that starts the generation. The clever generation window closes. When the generation process fails, a dialog informs about the error. Elsewise, the model validator window will display the newly generated configuration.

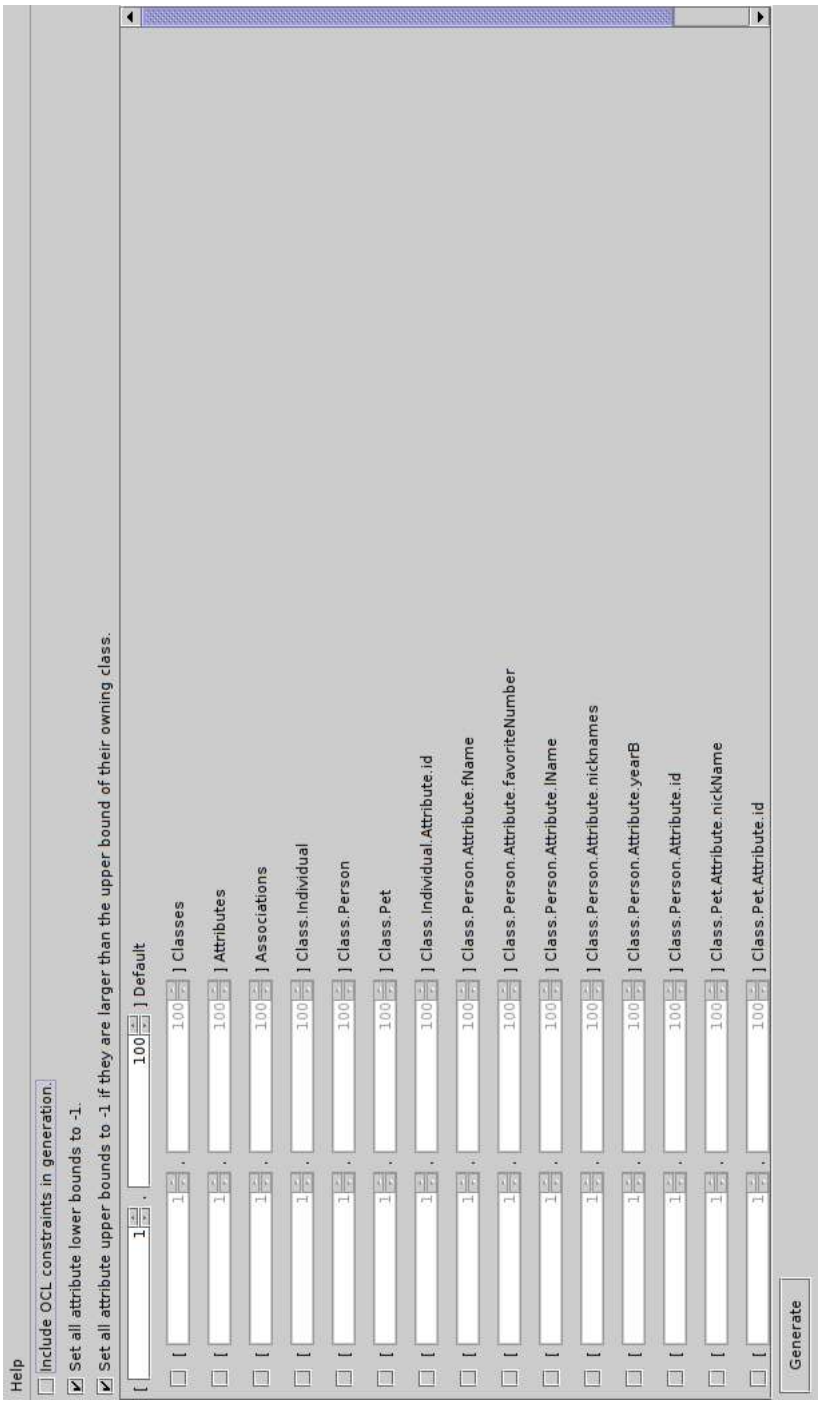

FIGURE 3.5: UML-based Specification Environment clever generation view for model from fig. 2.1

Figure 3.5 shows the initial clever generation window for the model from fig. 2.1. The number of model elements depends on the model. Therefor, there may be more model elements than their specification options can be displayed at the same time. The window is to small to display all specification options at once. Therefor, the window area containing the specification options is scrollable, which solves the problem.

3.3 Validation of configurations

Several problems are present with the permissive specification of configuration aspects. Validation of the absence of the corresponding problems may be a valuable feature. With the previously presented elimination of redundant search space also an adoption for validation already suggests itself. A configuration validation feature is implemented. The underlying problem and the benefits of this feature are presented in the following. The following sections contain details and examples of the implemented feature.

Configurations can specify inappropriate search spaces in terms of bounded model checking. Simple but inappropriate configuration details may cause the result of not finding any instance. The absence of certain kinds of inconsistencies could be validated automatically. Additionally, USE currently allows to specify search space aspects, that are obviously contradictory.

A feature is implemented that allows to analyse configurations for invalidities. Several validity rules are specified, which can be evaluated on all configurations. These rules are typified regarding the significance of the indicated invalidity. Furthermore, this is complemented with providing modification suggestions for rule violations. Also, the absence of redundant search space is defined as validity rule.

In the following subsections the implemented feature is presented in detail. The first section presents the adaption of the bounds tightening concept applied for the validation procedure. The general validation concepts are then presented in the second section. Here, the details of the underlying problems are not explained in advance but this introduction is implied in the presentation of the conceptualised validity rules. The feature supports proposing modifications for fixing invalidities. This is presented in the third section with the general application procedure of these fixes. The last two sections present the integrations in the UIs.

3.3.1 Constraint satisfaction problems applied for validation of instance finder configurations

The concept of bounds tightening can be applied to validate whether a configuration represents redundant search space. In the following, it is presented how this is done in detail.

As a prerequisite, several kinds of inconsistencies must not be present for successfully applying the procedure explained in the following. The inconsistencies, that must not be present, are described in the next section. Generally, the integration of the here presented validation aspects into the overall validation procedure, is included in the next section.

First of all, a copy is created for the to be validated configuration. Later on, the redundant search space will be eliminated using the copy and then the represented search spaces will be compared. The differences indicate presence of redundant search space.

Only the bounds regarding model elements, e.g. classes, attributes and associations, are of relevance for the bounds tightening procedure. Only those are processed in the copying process. Some aspects of the to be copied configuration are not suitable for the bounds tightening procedure and therefor must be replaced by appropriate values. The minimum and maximum numbers of objects are adopted for the copy. If the minimum number of defined attributes is -1 for an attribute, then the owning classes bounds are adopted. The minimum number of objects are used for the minimum number of defined attributes and the maximum number of objects

are used for the maximum number of defined attributes. If the minimum number of defined attributes is not -1 but the maximum number of defined attributes is -1, then the minimum number of defined attributes is adopted. As maximum number of defined attributes the larger value of the minimum number of defined attributes and the maximum number of objects of the owning class is used. Elsewise, the minimum and maximum numbers of defined attributes are adopted for the copy. The minimum numbers of links are adopted for the copy. If for an association the maximum number of links is -1, then the highest applicable integer number is used. Elsewise, the maximum number of links is adopted for the copy. The highest applicable integer number is 2147483646. This is highest integer value, representable in Java, minus one. The subtraction is done because the higher value results in errors with the used CSP framework.

Since the initial domain bounds specification in terms of CSP variables is possible specifically for all model elements, the configuration aspects contained in the copy can be used for the initial domain bounds specification. The minimum and maximum values used in all these bounds are computed. They are used as most general bounds in the initial domain bounds specification. With the initial domain bounds specification, a clever generation procedure is processed and the resulting configuration is further used. The model elements configuration aspects of the generated configuration and the original configuration are compared. If there are differing configuration aspects that represent a tighter search space with the generated configuration, then the original configuration contains redundant search space.

3.3.2 Validity rules

For validation of absence of inconsistencies in configurations, it suggests itself to define the unwanted inconsistencies. This is done here with a definition of 32 rules. For the procedure of evaluating the rules, there is the special aspect that some rules depend on no violations of specific other rules. It is specified for each rule on which other rules it depends on. Since one rule can depend on inviolations of two other rules, which both depend on inviolation of one same rule, the dependency structurally is a representable as a graph. For deterministic terminating rule evaluating procedure, there must be no loops in the dependency structure.

Rule	Definition overview
1	Integer type settings must be conditionally enabled.
2	Integer type settings must be conditionally disabled.
3	Minimum integer value must be less than or equal maximum.
4	Preferred integer values must not contain invalid values.
5	All rules regarding integer type settings must be inviolated.
6	String type settings must be conditionally enabled.
7	String type settings must be conditionally disabled.
8	Minimum and maximum number of strings must be greater than 0.
9	Minimum number of strings must be less than or equal maximum.
10	Preferred string values must not contain to much values.
11	All rules regarding string type settings must be inviolated.
12	Real type settings must be conditionally enabled.
13	Real type settings must be conditionally disabled.
14	Minimum real value must be less than or equal maximum.
15	Real step value must be less than or equal difference of maximum and minimum.
16	Maximum real value must be reachable from minimum in steps of real step value.
17	Preferred real values must not contain invalid values.
18	All rules regarding real type settings must be inviolated.
19	All rules regarding type settings must be inviolated.
20	Minimum and maximum number of objects must be greater than or equal 0.
21	Minimum number of objects must be greater less or equal maximum.
22	Preferred objects must not contain too much values.
23	Minimum and maximum number of objects of abstract classes must be sums of directly derived classes.
24	Maximum number of links must be greater than or equal -1.
25	Minimum number of links must be greater than or equal 0.
26	Maximum number of links must be greater than or equal minimum if not -1.
27	Required links must not contain invalid object names.
28	Minimum and maximum number of links must be greater than or equal number of required links.
29	All rules regarding association settings must be inviolated.
30	All rules regarding class settings must be inviolated.
31	Minimum number of defined attributes must be greater than or equal -1.
32	Maximum number of defined attributes must be greater than or equal -1.
33	Minimum number of defined attributes must be less than or equal maximum if not -1.
34	Maximum number of defined attributes must be less than or equal maximum number of objects if not -1.
35	Minimum number of contained attributes for collection type attributes must be greater than or equal 0.
36	Maximum number of contained attributes for collection type attributes must be greater than or equal minimum or -1.
37	All rules regarding attribute settings must be inviolated.
38	Invariants must be unnegated if not active.
39	All rules regarding invariant settings must be inviolated.
40	The configuration must not represent search space that contains redundant search space.
41	All other rules must be inviolated.

TABLE 3.5: Validity rules overview

Rule	Inviolation dependencies	Definition
1	$\emptyset$	This is violated when integer type settings are disabled but should be enabled. They should be enabled if there is at least one class that is configured to potentially have instances and that has at least one attribute of type integer that is configured to potentially be defined.
2	$\emptyset$	This is violated when integer type settings are enabled but should be disabled. They should be disabled if there is not at least one class that is configured to potentially have instances and that has at least one attribute of type integer that is configured to potentially be defined.
3	$\{1,2\}$	This is violated if integer type settings are enabled and the minimum integer number is not less than or equal the maximum integer number.
4	$\{3\}$	This is violated if integer type settings are enabled and the not all preferred integer values are contained in the domain defined by minimum and maximum integer number.
5	$\{4\}$	Is only not violated if implicitly all rules regarding configuration aspects for integer type are not violated.
6	$\emptyset$	This is violated when string type settings are disabled but should be enabled. They should be enabled if there is at least one class that is configured to potentially have instances and that has at least one attribute of type string that is configured to potentially be defined.
7	$\emptyset$	This is violated when string type settings are enabled but should be disabled. They should be disabled if there is not at least one class that is configured to potentially have instances and that has at least one attribute of type sting that is configured to potentially be defined.
8	$\{6,7\}$	This is violated if string type settings are enabled and the minimum and maximum number of string values is less than or equal 0.
9	$\{8\}$	This is violated if string type settings are enabled and the minimum number of string values is not less than or equal the maximum number of string values.
10	$\{8\}$	This is violated if string type settings are enabled and the preferred values contain more elements than the maximum number of string values specifies.
11	$\{9,10\}$	Is only not violated if implicitly all rules regarding configuration aspects for string type are not violated.
12	$\emptyset$	This is violated when real type settings are disabled but should be enabled. They should be enabled if there is at least one class that is configured to potentially have instances and that has at least one attribute of type real that is configured to potentially be defined.

TABLE 3.6: Validity rules (Part 1)

Rule	Inviolation dependencies	Definition
13	$\emptyset$	This is violated when real type settings are enabled but should be disabled. They should be disabled if there is not at least one class that is configured to potentially have instances and that has at least one attribute of type real that is configured to potentially be defined.
14	$\{12, 13\}$	This is violated if real type settings are enabled and the minimum real number is not less than or equal the maximum real number.
15	$\{14\}$	This is violated if real type settings are enabled and the real step value is not less than or equal the difference of maximum and minimum real number.
16	$\{15\}$	This is violated if real type settings are enabled and the modulo of difference of maximum and minimum real and the real step value is not 0. This means the maximum real number must be reachable from the minimum real number in steps of the real step value.
17	$\{16\}$	This is violated if real type settings are enabled and not all preferred real values are contained in the domain defined by minimum and maximum real numbers and the real step value.
18	$\{17\}$	Is only not violated if implicitly all rules regarding configuration aspects for real type are not violated.
19	$\{5, 11, 18\}$	Is only not violated if implicitly all rules regarding all configuration aspects for types are not violated.
20	$\emptyset$	This is violated if for at least one class the minimum or maximum number of objects is not greater than or equal 0.
21	$\{20\}$	This is violated if for at least one class the minimum number of objects is not less than or equal the maximum number of objects.
22	$\{21\}$	This is violated if for at least one class the preferred values contain more elements than the maximum number of objects specifies.
23	$\{21\}$	This is violated if for at least one abstract class the minimum number of objects is not equal to the sum of the minimum numbers of objects of directly derived classes or the maximum number of objects is not equal to the sum of the maximum numbers of objects of directly derived classes.
24	$\emptyset$	This is violated if for at least one association the maximum number of links is not greater than or equal to -1.
25	$\emptyset$	This is violated if for at least one association the minimum number of links is not greater than or equal to 0.
26	$\{24, 25\}$	This is violated if for at least one association the maximum number of links is not -1 and not greater than or equal to the minimum number of links.

TABLE 3.7: Validity rules (Part 2)

Rule	Inviolation dependencies	Definition
27	$\{26\}$	This is violated if not all object names contained in the preferred links are contained in the preferred object names.
28	$\{27\}$	This is violated if for at least one association the number of required links is less than or equal to the minimum and maximum number of links.
29	$\{28\}$	Is only not violated if implicitly all rules regarding configuration aspects for associations are not violated.
30	$\{22, 23\}$	Is only not violated if implicitly all rules regarding configuration aspects for classes are not violated.
31	$\emptyset$	This is violated if for at least one attribute the minimum number of defined attributes is not greater than or equal to -1.
32	$\emptyset$	This is violated if for at least one attribute the maximum number of defined attributes is not greater than or equal to -1.
33	$\{31, 32\}$	This is violated if for at least one attribute the minimum number of defined attributes is not -1 and is also not less than or equal to the maximum number of defined attributes.
34	$\{30, 33\}$	This is violated if for at least one attribute the maximum is not -1 and not less than or equal the number of objects of the owning class.
35	$\emptyset$	This is violated if for at least one attribute of collection type the minimum number of contained elements in defined attributes is not greater than or equal to 0.
36	$\{35\}$	This is violated if for at least one attribute of collection type the maximum number of contained elements in defined attributes is not -1 and not greater than or equal to the minimum number of contained elements in defined attributes.
37	$\{34, 36\}$	Is only not violated if implicitly all rules regarding configuration aspects for classes and their attributes are not violated.
38	$\emptyset$	This is violated if for at one invariant is configured to be negated while it is not configured to be activated.
39	$\{38\}$	Is only not violated if implicitly all rules regarding configuration aspects for invariants are not violated.
40	$\{30, 37, 29\}$	This is violated if the model elements configuration aspects represent a search space containing redundant search space.
41	$\{19, 30, 37, 29, 39, 40\}$	Is only not violated if implicitly all rules are not violated.

TABLE 3.8: Validity rules (Part 3)

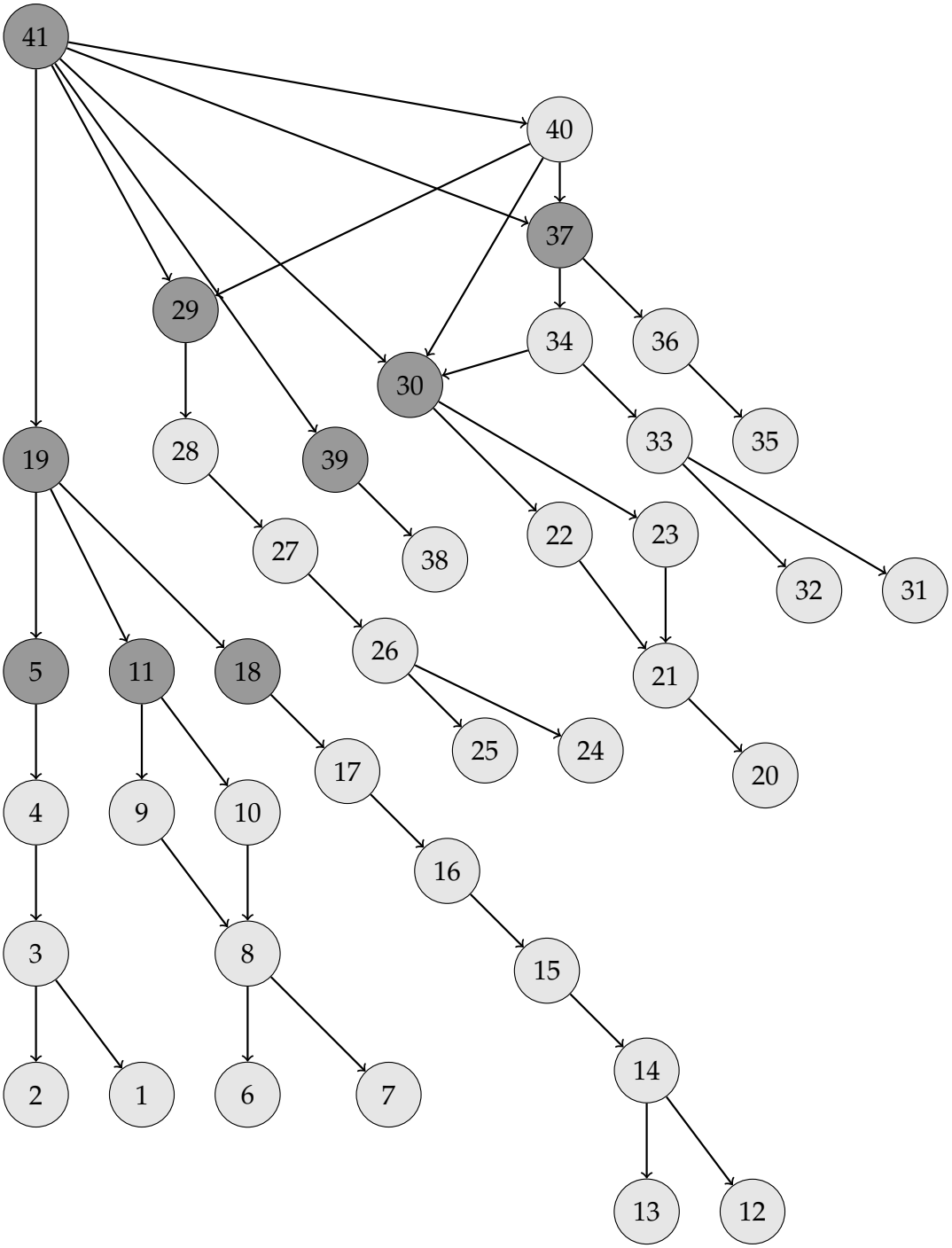

FIGURE 3.6: Inconsistency dependency structure for validity rules

The set of validity rules is presented in table 3.6, 3.7 and 3.8. 41 rules are defined, containing 32 rules that have an explicit validity definition. The other rules are just containers for subsets of the rules.

Figure 3.6 shows the evaluation dependency structure of the validity rules. Rules without an explicit validity definition are visualised with a darker background. Obviously there are no loops. The dependency structure also does only show finite chains of dependency relations. Therefor, the rule evaluating procedure can be deterministic and terminating.

When evaluating a validity rule, firstly all validity rules the rule depends on are evaluated. If they are all not violated and there is a specific validity definition for the rule, it is evaluated whether it is not violated. If they are all not violated and there is no specific validity definition, the rule is not violated. The overall validation is achieved by starting with rule 41. Implicitly, all other rules are not allowed to be violated.

A rule can be violated more than once, when validating one configuration. If the model contains multiple non abstract classes, then for example rule 20 can be violated for each class. This provides an extended basis for proposing applicable fixes for violations. When evaluating a rule, not only the first violation but all violations of the rule are computed.

3.3.3 Proposed applicable fixes

For violations of validity rules, there may are some potential modifications on the relevant configuration aspects, that bring absence of the violations. The computation of proposed applicable modifications, that (partially) fix the relevant configuration aspects, is implemented. This proposed fixes are presented in the following. The modifications each depends on the corresponding validity rules.

Violated rule	Proposed applicable fixes
1	Multiple fixes are provided: • Enable integer type settings. • For each integer attribute whose maximum defined value is greater than 0 and whose classes maximum number of instances is not 0, set minimum and maximum number of its owning class to 0. • For each integer attribute whose maximum defined value is greater than 0 and whose classes maximum number of instances is not 0, set minimum and maximum number of defined attributes to 0.
2	Disable integer type settings.
3	Multiple fixes are provided: • Set minimum integer value to value of maximum integer value. • Set maximum integer value to value of minimum integer value.
4	Multiple fixes are provided: • Remove all values from the preferred integer values that are not contained in the domain represented by minimum and maximum integer values. • Set minimum integer value to lowest value contained in preferred integer values if this value is lower than the original minimum integer value. • Set maximum integer value to highest value contained in preferred integer values if this value is higher than the original maximum integer value. • If both foregoing fixes are provided, they are also provided as bundle.

TABLE 3.9: Proposed applicable fixes for validity rules (Part 1)

Violated rule	Proposed applicable fixes
6	Multiple fixes are provided: • Enable string type settings. • For each string attribute whose maximum defined value is greater than 0 and whose classes maximum number of instances is not 0, set minimum and maximum number of its owning class to 0. • For each string attribute whose maximum defined value is greater than 0 and whose classes maximum number of instances is not 0, set minimum and maximum number of defined attributes to 0.
7	Disable string type settings.
8	Multiple fixes are provided: • If minimum and maximum numbers of different string values are less than 0, set both to 0. • If minimum number of different string values is less than 0, set it to 0. • If maximum number of different string values is less than 0, set it to 0.
9	Multiple fixes are provided: • Set minimum number of different string values to value of maximum number of different string values. • Set maximum number of different string values to value of minimum number of different string values.
10	Multiple fixes are provided: • Set maximum number of different string values to size of preferred string values. • Remove the last preferred string values so that the size is equal to the maximum number of different string values.
12	Multiple fixes are provided: • Enable real type settings. • For each real attribute which maximum defined value greater than 0 and whose classes maximum number of instances is not 0 set minimum and maximum number of its owning class to 0. • For each real attribute which maximum defined value greater than 0 and whose classes maximum number of instances is not 0 set minimum and maximum number of defined attributes to 0.
13	Disable real type settings.
14	Multiple fixes are provided: • Set minimum real value to value of maximum real value. • Set maximum real value to value of minimum real value.
15	Multiple fixes are provided: • If real step value is less than 0, set it to the difference of maximum and minimum real values. • If real step value is greater than the difference of maximum and minimum real values, set the maximum real value to the sum of the difference and the minimum real value. • If real step value is greater than the difference of maximum and minimum real values, set the minimum real value to the difference of the maximum real value and the other difference.

TABLE 3.10: Proposed applicable fixes for validity rules (Part 2)

Violated rule	Proposed applicable fixes
16	Multiple fixes are provided: • If there exists a lower value than the maximum real value for that modulo of difference of this value and minimum real and the real step value is 0 and this value is greater than the minimum real value, set the maximum real value to this value. • Set the maximum real value to the next higher value for that modulo of difference of this value and minimum real and the real step value is 0. • Set the real step value to the difference of the maximum and minimum real value.
17	Multiple fixes are provided: • Set minimum real value to lowest value contained in preferred real values if this value is lower than the original minimum real value. • Remove all values from the preferred real values that are lower than the minimum real value. • Set maximum real value to highest value contained in preferred real values if this value is higher than the original maximum real value. • Remove all values from the preferred real values that are higher than the maximum real value. • Remove all values from the preferred real values that are greater than or equal to the minimum real value and less than or equal to the maximum real value but are not contained in the domain represented by minimum and maximum real value and the real step value. • Remove all values from the preferred real values that are not contained in the domain represented by minimum and maximum real value and the real step value.
20	Multiple fixes are provided for each class this is violated for: • If the minimum and maximum numbers of objects is less than 0, set both to 0. • If the minimum number of objects is less than 0, set it to 0. • If the maximum number of objects is less than 0, set it to 0.
21	Multiple fixes are provided for each class this is violated for: • Set minimum number of objects to value of maximum number of objects. • Set maximum number of objects to value of minimum number of objects.
22	Multiple fixes are provided for each class this is violated for: • Set maximum number objects to size of preferred object names. • Remove the last preferred objects names so that the size is equal to the maximum number of objects.

TABLE 3.11: Proposed applicable fixes for validity rules (Part 3)

Violated rule	Proposed applicable fixes
23	Multiple fixes are provided for each class this is violated for: • If the minimum number of objects is greater than the sum of the minimum numbers of objects of the derived classes and the maximum number of objects is less than the sum of the maximum numbers of objects of the derived classed set both to the corresponding sum. • If the minimum number of objects is greater than the sum of the minimum numbers of objects of the derived classes set the minimum number of objects to the sum. • If the maximum number of objects is less than the sum of the maximum numbers of objects of the derived classed set the maximum number of objects to the sum.
24	Multiple fixes are provided for each association this is violated for: • If the maximum number of links is less than -1, set it to -1. • If the maximum number of links is less than -1, set it to 0.
25	If the minimum number of links is less than 0, set it to 0.
26	Multiple fixes are provided for each association this is violated for: If the maximum number of links is not -1 and the minimum number of links is greater than the maximum number of links: • Set the minimum number of links to 0. • Set the maximum number of links to -1. • Set the minimum number of links to the maximum number of links. • Set the maximum number of links to the minimum number of links.
27	Multiple fixes are provided for each association this is violated for: • Remove all required links containing instance names that are not contained in the preferred instance names of the corresponding classes. • If there are ten or less links to be removed, provide a removal of each. • For each class the preferred instance names do not contain corresponding instance names from the required links add the missing instance names. • If there are more than one class with the foregoing fix, provide also a bundled fix of those.
28	Multiple fixes are provided for each association this is violated for: • If minimum and maximum numbers of links are less than the size of required links, set both to the size of required links. • If minimum number of links is less than the size of required links, set the minimum number of links to the size of required links. • If maximum number of links is less than the size of required links, set the maximum number of links to the size of required links.
31	Multiple fixes are provided for each attribute this is violated for: If the minimum number of defined attributes is less than -1: • Set it to -1. • Set it to 0.

TABLE 3.12: Proposed applicable fixes for validity rules (Part 4)

Violated rule	Proposed applicable fixes
32	Multiple fixes are provided for each attribute this is violated for: If the maximum number of defined attributes is less than -1: • Set it to -1. • Set it to 0.
33	Multiple fixes are provided for each attribute this is violated for: If the maximum number of defined attributes is not than -1 and the minimum number of defined attributes is greater than the maximum number of defined attributes: • Set the minimum number of defined attributes to 0. • Set the maximum number of defined attributes to -1. • Set the minimum number of defined attributes to the maximum number of defined attributes. • Set the maximum number of defined attributes to the minimum number of defined attributes.
34	Multiple fixes are provided for each attribute this is violated for: If the maximum number of defined attributes is greater than the maximum number of objects of the owning class: • Set the maximum number of defined attributes to 0. • Set the maximum number of defined attributes to the maximum number of objects of the owning class. • Set the maximum number of defined attributes to -1. • Set the maximum number of objects of the owning class to the maximum number of defined attributes.
35	Set the minimum number of contained elements to 0.
36	Multiple fixes are provided for each attribute this is violated for: • Set the maximum number of contained elements to the minimum number of contained elements. • Set the maximum number of contained elements to -1.
38	Multiple fixes are provided for each invariant this is violated for: • Set the invariant activated. • Set the invariant unnegated.

TABLE 3.13: Proposed applicable fixes for validity rules (Part 5)

Violated rule	Proposed applicable fixes
40	Multiple fixes are provided for each model element this is violated for: • For classes: – If minimum number of objects is less than the optimised minimum number of objects, set it to the optimised value. – If maximum number of objects is greater than the optimised maximum number of objects, set it to the optimised value. – If minimum number of objects is less than the optimised minimum number of objects and the maximum number of objects is greater than the optimised maximum number of objects, set both to their corresponding optimised value. • For attributes: – If minimum number of defined attributes is less than the optimised minimum number of defined attributes, set it to the optimised value. – If maximum number of defined attributes is greater than the optimised maximum number of defined attributes, set it to the optimised value. – If minimum number of defined attributes is less than the optimised minimum number of defined attributes and the maximum number of defined attributes is greater than the optimised maximum number of defined attributes, set both to their corresponding optimised value. • For associations: – If minimum number of links is less than the optimised minimum number of links, set it to the optimised value. – If maximum number of links is greater than the optimised maximum number of links, set it to the optimised value. – If minimum number of links is less than the optimised minimum number of links and the maximum number of links is greater than the optimised maximum number of links, set both to their corresponding optimised value.

TABLE 3.14: Proposed applicable fixes for validity rules (Part 6)

The set of proposed applicable fixes for violated rules is presented with table 3.9 to 3.14. Fixes are only computed for the 32 rules, that have an explicit validity definition. There is always is a fix proposed for each rule, that brings absence of the violation. Since a rule may be violated more than once, not all violations of the rule may be absent after applying the fix. There may also be fixes for some violations, that do not bring absence of the violation. For example, if rule 4 is violated and all the fixes are provided but only the second fix is applied, the rule will be still violated. This is the case, when the preferred integer values contain values that are less than the minimum integer value and values that are greater than the maximum integer value. After applying the second fix with setting the minimum integer value to the lowest value contained in the preferred integer values, the preferred integer values still contain values that are greater than the maximum integer value. Therefor, the same rule is still violated after applying the second fix. A differentiation is given with so called full fixes and partial fixes. Applying full fixes resolves a violation but

partial fixes do not resolve violations.

Applying fixes may cause other rules to be violated. This can occur when a violation is fixed and then the rules are evaluated that are not evaluated before because the evaluation depends on inviolation of the before violated rule. But also when applying fixes other rules may become violated, whose evaluation does not depend on inviolation of the fixed violated rule. For example, if rule 40 is violated so that the third or second fix is applied, then rule 34 may become violated. When modifying the maximum number of objects for a class, also the valid maximum numbers of defined attributes for all attributes of the class change implicitly. If there is a value for an attribute of the class that is greater than the new maximum of objects, then rule 34 will be violated. The process of applying rules must be iteratively and also user driven, because the fixes to be applied must be selected by the user.

Generally the concept of the procedure of applying fixes is the same for both UIs. Firstly, all violations and fixes are presented. The user then may select a fix to apply but also has the option to not apply any fix. After applying a fix the procedure optionally starts from the beginning. In the whole procedure only a cached version of the configuration may be edited. The option of replacing the original configuration with the modified configuration will be given later. This differs for the different UIs.

3.3.4 Command-line interface integration

The presented validation concepts are implemented in USE. It extends the functionality of the USE Shell, the CLI. This is explained in the following.

Prerequisite for using the functionality is an UML/OCL model loaded in USE. The validation command is available as `modelvalidator -validateConfig` and `mv -vc`. As mandatory argument, the path to a configuration file must be specified. The respected configurations from the file can be optionally selected by specifying the names as arguments. When no (valid) names are specified, all configurations from the file are validated. Specified names are validated before validation. When a name is specified twice it will be only used once in the validation process. If a name is specified that is not contained in the file, it is ignored. In both cases an informative output is given. When (valid) names are specified only the corresponding configurations from the file are validated. All violations of validity rules are presented textually. There may be multiple violations for some rules. For each violation, the fixes are also presented textually. The command for validation with interactive fixing is available as `modelvalidator -fixConfig` and `mv -fc`. As mandatory argument, the path to a configuration file must be specified. A valid name of a configuration contained in the file must also be specified. The corresponding configuration from the file is validated. All violations of validity rules are presented textually. There may be multiple violations for some rules. For each violation, the fixes are also presented textually. A number, which is unique per fixing iteration, is given for each proposed fix. After the textual presentation of the validation result is given, one can give the number of a fix as an input or exit the validation process with or without saving the modified configuration in its file. For the last two options, also unique constant numbers are given.

```
1 [config1]
2
3 Integer_min = 10
4 Integer_max = -10
```

LISTING 3.5: UML-based Specification Environment instance finder configuration for model from fig. 2.1

```
1  Config config1 is invalid.
2      Rule INTEGER_SETTINGS_MIN_LESS_THAN_OR_EQUAL_MAX is violated.
3          Integer_min (10) should be less than or equal Integer_max (-10)
               .
4              To get closer to validity perform one of the following
                   fixes:
5                  [fF] | Set Integer_min to -10.
6                  [fF] | Set Integer_max to 10.
7      Rule STRING_SETTINGS_NESSESSARY is violated.
8          Settings for String should be enabled. There is at least one
               class which have an attribute of type String and is
               configured to be able to have instances and to have values
               for that attribute.
9              To get closer to validity perform one of the following
                   fixes:
10                 [fF] | Enable settings for String.
11                 [pF] | Set Individual_min to 0 and Individual_max to 0.
12                 [pF] | Set Individual_id_min to 0 and Individual_id_max
                        to 0.
13                 [pF] | Set Pet_min to 0 and Pet_max to 0.
14                 [pF] | Set Pet_id_min to 0 and Pet_id_max to 0.
15                 [pF] | Set Pet_nickName_min to 0 and Pet_nickName_max
                       to 0.
16                 [pF] | Set Person_min to 0 and Person_max to 0.
17                 [pF] | Set Person_lName_min to 0 and Person_lName_max
                       to 0.
18                 [pF] | Set Person_id_min to 0 and Person_id_max to 0.
19                 [pF] | Set Person_fName_min to 0 and Person_fName_max
                       to 0.
20     Rule REAL_SETTINGS_NESSESSARY is violated.
21         Settings for Real should be enabled. There is at least one
               class which have an attribute of type Real and is
               configured to be able to have instances and to have values
               for that attribute.
22             To get closer to validity perform one of the following
                   fixes:
23                 [fF] | Enable settings for Real.
24                 [fF] | Set Person_min to 0 and Person_max to 0.
25                 [fF] | Set Person_favoriteNumber_min to 0 and
                           Person_favoriteNumber_max to 0.
26     Rule
           CLASS_SETTINGS_ABSTRACT_CLASS_BOUNDS_EQUALS_SUM_OF_BOUNDS_OF_DERIVED_CLASSES
            is violated.
27         Individual_min (1) should be less than or equal 2 and
               Individual_max (1) should be greater than or equal 2
               because the lower bound of an abstract Class needs to be
               less than or equal the sum of the lower bounds of derived
               Classes and the upper bound needs to be greater than or
               equal the sum of the upper bounds of the derived Classes (
               Pet,Person).
28             To get closer to validity perform the following fix:
29                 [fF] | Set Individual_max to 2.
```

LISTING 3.6: UML-based Specification Environment command-line interface output for validation of configuration from listing 3.5

Listing 3.6 shows the output for validation of the configuration from listing 3.5. Obviously, four rules are violated with this configuration. The explicitly given integer minimum and maximum value are not valid, because the minimum value is greater than the maximum value. String type and real type settings are not enabled, because no configuration aspects are specified for that. But they need to be enabled,

because there is the default value 1 used for at least class `Person` and -1 as minimum number of defined attributes for all attributes of the class. This class has at least one attribute of both types. The corresponding type settings must be given. While there is no option to apply the fixes in this paradigm, the presentation of the fixes givens crucial information.

```
1  [config1]
2
3  Integer_min = 10
4  Integer_max = -10
5
6  String_max = 10
7
8  Real_max = 10.0
9
10 Pet_min = 2
11 Pet_max = 4
12
13 Person_max = 4
14
15 Individual_max = 8
16
17 Parenthood_max = -1
18
19 PetSitting_max = -1
```

LISTING 3.7: UML-based Specification Environment textual instance finder configurations for model from fig. 2.1 used for fixing of invalidities

```
1  Validation results:
2
3  Config config1 is invalid.
4      Rule INTEGER_SETTINGS_MIN_LESS_THAN_OR_EQUAL_MAX is violated.
5          Integer_min (10) should be less than or equal Integer_max (-10)
               .
6              To get closer to validity perform one of the following
                   fixes:
7  (1)             [fF] | Set Integer_min to -10.
8  (2)             [fF] | Set Integer_max to 10.
9      Rule SETTINGS_UNTIGHTENABLE_BOUNDS is violated.
10         Tighter bounds are computed for class Individual based on the
               other settings.
11             To get closer to validity perform one of the following
                   fixes:
12 (3)             [pF] | Set Individual_min to 3.
13 (4)             [pF] | Set Individual_max to 6.
14 (5)             [fF] | Set Individual_min to 3 and Individual_max to 6.
15         Tighter bounds are computed for class Person based on the other
               settings.
16             To get closer to validity perform the following fix:
17 (6)             [fF] | Set Person_max to 2.
18         Tighter bounds are computed for association PetSitting based on
               the other settings.
19             To get closer to validity perform the following fix:
20 (7)             [fF] | Set PetSitting_min to 2.
21
22 Please choose a fix to apply by its number
23  or type "0" to save the configuration in the current state and
       terminate this fixing process
24 or "-1" to terminate this fixing process without saving.
25 option: 2
```

```
26
27 Validation results:
28
29 Config config1 is invalid.
30     Rule SETTINGS_UNTIGHTENABLE_BOUNDS is violated.
31         Tighter bounds are computed for class Individual based on the
               other settings.
32             To get closer to validity perform one of the following
                   fixes:
33 (1)             [pF] | Set Individual_min to 3.
34 (2)             [pF] | Set Individual_max to 6.
35 (3)             [fF] | Set Individual_min to 3 and Individual_max to 6.
36         Tighter bounds are computed for class Person based on the other
               settings.
37             To get closer to validity perform the following fix:
38 (4)             [fF] | Set Person_max to 2.
39         Tighter bounds are computed for association PetSitting based on
               the other settings.
40             To get closer to validity perform the following fix:
41 (5)             [fF] | Set PetSitting_min to 2.
42
43 Please choose a fix to apply by its number
44  or type "0" to save the configuration in the current state and
       terminate this fixing process
45 or "-1" to terminate this fixing process without saving.
46 option: 5
47
48 Validation results:
49
50 Config config1 is invalid.
51     Rule SETTINGS_UNTIGHTENABLE_BOUNDS is violated.
52         Tighter bounds are computed for class Individual based on the
               other settings.
53             To get closer to validity perform one of the following
                   fixes:
54 (1)             [pF] | Set Individual_min to 3.
55 (2)             [pF] | Set Individual_max to 6.
56 (3)             [fF] | Set Individual_min to 3 and Individual_max to 6.
57         Tighter bounds are computed for class Person based on the other
               settings.
58             To get closer to validity perform the following fix:
59 (4)             [fF] | Set Person_max to 2.
60
61 Please choose a fix to apply by its number
62  or type "0" to save the configuration in the current state and
       terminate this fixing process
63 or "-1" to terminate this fixing process without saving.
64 option: 4
65
66 Validation results:
67
68 Config config1 is invalid.
69     Rule SETTINGS_UNTIGHTENABLE_BOUNDS is violated.
70         Tighter bounds are computed for class Individual based on the
               other settings.
71             To get closer to validity perform one of the following
                   fixes:
72 (1)             [pF] | Set Individual_min to 3.
73 (2)             [pF] | Set Individual_max to 6.
74 (3)             [fF] | Set Individual_min to 3 and Individual_max to 6.
75
76 Please choose a fix to apply by its number
```

```
77  or type "0" to save the configuration in the current state and
        terminate this fixing process
78  or "-1" to terminate this fixing process without saving.
79  option: 3
80
81  Validation results:
82
83  Config config1 is valid.
84
85
86  Please type "0" to save the configuration in the current state and
        terminate this fixing process
87  or "-1" to terminate this fixing process without saving.
88  option: 0
```

LISTING 3.8: UML-based Specification Environment command-line interface output for fixing of invalidities of configuration from listing 3.7

```
1   [config1]
2
3   Integer_min = 10
4   Integer_max = 10
5
6   String_max = 10
7
8   Real_min = -2.0
9   Real_max = 10.0
10  Real_step = 0.5
11
12  # ------------------------------------------------------------------------
        Individual
13
14  Individual_id_min = -1
15  Individual_id_max = -1
16
17  # ------------------------------------------------------------------------
        Person
18  Person_min = 1
19  Person_max = 2
20
21  Person_fName_min = -1
22  Person_fName_max = -1
23  Person_favoriteNumber_min = -1
24  Person_favoriteNumber_max = -1
25  Person_lName_min = -1
26  Person_lName_max = -1
27  Person_nicknames_min = -1
28  Person_nicknames_max = -1
29  Person_nicknames_minSize = 0
30  Person_nicknames_maxSize = -1
31  Person_yearB_min = -1
32  Person_yearB_max = -1
33
34  # Parenthood (parent:Person, child:Person) - - - - - - - - - - - - - - - -
        - - - - -
35  Parenthood_min = 1
36  Parenthood_max = -1
37
38  # PetSitting (sitter:Person, pet:Pet) - - - - - - - - - - - - - - - - - -
        - - - - -
39  PetSitting_min = 2
```

```
40 PetSitting_max = -1
41
42 #
     ---------------------------------------------------------------------------
     Pet
43 Pet_min = 2
44 Pet_max = 4
45
46 Pet_nickName_min = -1
47 Pet_nickName_max = -1
48 #
     ---------------------------------------------------------------------------

49 aggregationcyclefreeness = on
50 forbiddensharing = on
```

LISTING 3.9: UML-based Specification Environment textual instance finder configurations for model from fig. 2.1 resulting with fixing of invalidities (listing 3.8)

Listing 3.8 shows the output for fixing the violations of validity rules from listing 3.7. The configuration aspects for integer values are here also not valid. Firstly this is fixed by setting the integer maximum value to 10. Then no other rules than rule 40 are violated. There are tighter bounds computed for the two non abstract classes. Since bounds for abstract classes with deriving classes can be derived from the deriving classes, in the second and third step of the interactive fixing process the proposed fixes are applied for the bounds of the two non abstract classes. In the fourth step the proposed bounds for the abstract class are applied. The configuration is then saved, which results in the configuration presented in listing 3.9.

3.3.5 Graphical user interface integration

The validation functionality is also integrated in the GUI. This is explained in the following and then complemented by adapting the example from the previous section.

Prerequisites for using the functionality are an UML/OCL model loaded in USE and an existing model validator configuration for the model loaded with the model validator plugin. The comparison command is available as menu item `Check`, in menu `Configuration`, in the menu bar at the top of the model validator window. Clicking the menu item opens a validation window. The model validator window will freeze until the validation window is closed. Initially the configuration from the state in the model validator window is validated. While the configuration may have been loaded from a file, it can contain unsaved changes. The state with changes is used for validation.

All violations of validity rules are presented textually in the validation window. Their information are grouped by a frame. There may be multiple violations for some rules. For each violation the fixes are also presented textually. A button for applying the fix is given for each fix. On the bottom of the window an initially checked checkbox is given. When it is checked, the window will close after applying a fix. Elsewise, the window will reopen and the validation results for the modified configuration are given. When it is computed that there are no invalidities, the validation window does not open or reopen but a dialog informs about this result.

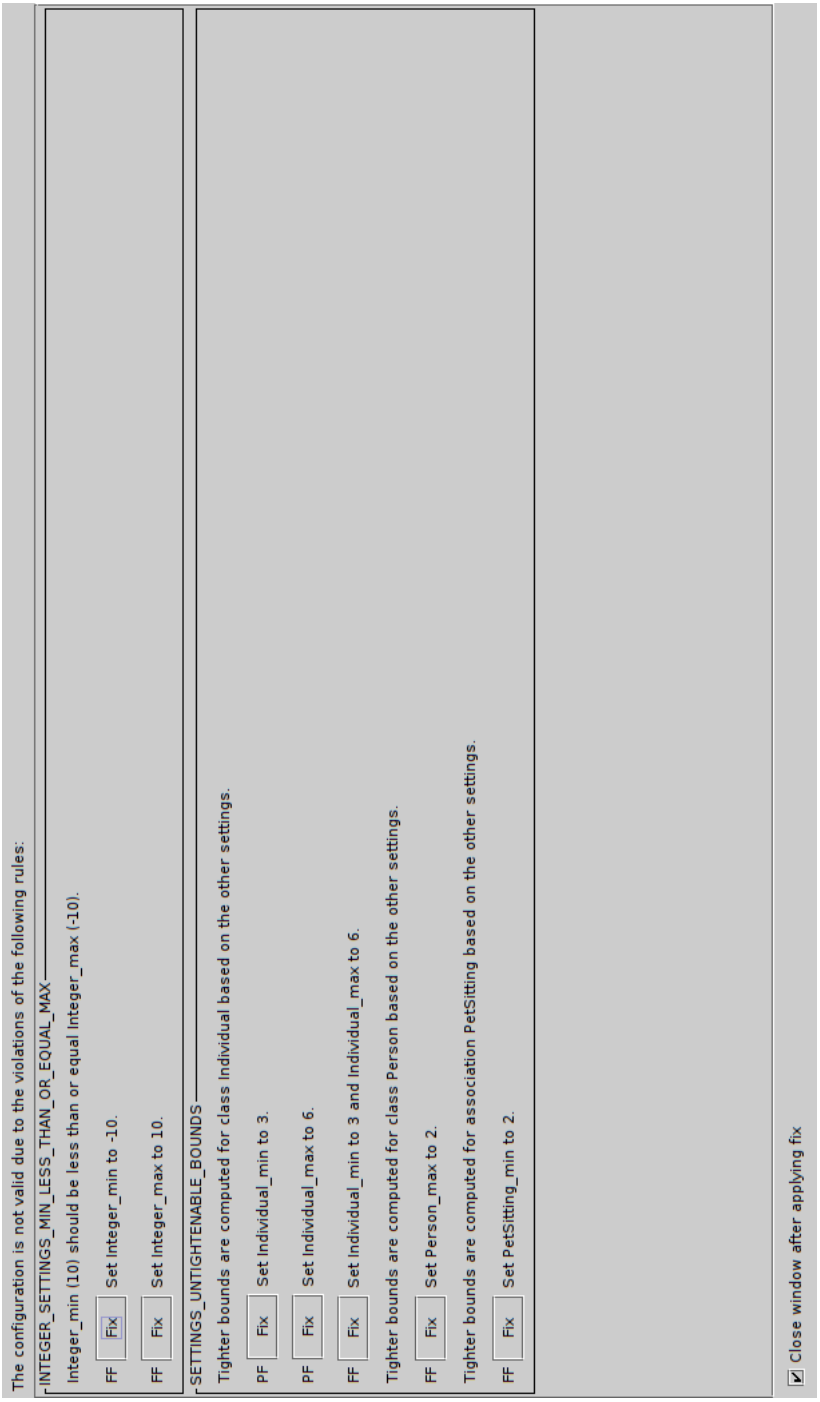

FIGURE 3.7: UML-based Specification Environment validation overview for validation of configuration from listing 3.7

Figure 3.7 shows the validation window for validation of the configuration from listing 3.7. Here, an equivalent fixing process to the process documented in listing 3.8 can be done. The configuration can be saved by using the functions from the model validator window.

Chapter 4

Evaluation

Three implemented features for complementing workflows regarding validation and verification of UML class diagrams with USE are presented previously. Since the application of the features have also disadvantages, it is for example associated with more effort for users, it should be highlighted that there are also benefits. This chapter presents the evaluation of crucial aspects of only one of the features. After that, other ideas on not performed evaluation are also presented.

The instance finder can be utilised for several validation and verification use cases. For each the suitability of the instance finder configuration is crucial. Configurations can represent redundant search space. Redundant search space may already be absent for configurations generated with the clever generation feature. Comparing instance finding processes execution times for processes with simple configurations and clever configurations gives information about whether the feature reduces time expenditures. For meaningful results the clever configurations must be generated on the basis of the simple configurations. A wide range of models and forms of configurations for each model must also be considered. The evaluation whether the clever generation feature reduces instance finding execution times is presented in the following. Data is generated based on a wide set of configurations and random models. A similar evaluation is done by Clarisó, González, and Cabot (2019).

The clever generation procedure will always fail for models if the instance finder can not find instances with the corresponding simple configuration. Also, the clever generation procedure will always fail for uninstantiable models. For such models, the time needed for the clever generation procedure to fail can be compared with the time needed to proof uninstantiability in context of the corresponding simple configuration. For generally instanticiable models, the time needed for instance finding with a simple configuration can be compared with the sum of the times needed for generating the corresponding clever configuration and executing instance finding with this. Therefor, two sets of random models are needed. One with only generally instanticiable models and one with only uninstanticiable models are needed.

For the generated models and configurations only some model elements are relevant. Since invariants are not respected in clever generation only classes and associations are relevant. Multiplicities of associations and whether classes are abstract or derived are also relevant. Model and configuration aspects for invariants, attributes and types can be ignored. This determines the parameters of the random model generation procedure.

Seven parameters are used for the random model generation procedure. Firstly A is the probability of each pair of classes to be part of binary associations. Secondly C is the number of classes. CA is the probability of each class to be abstract. CD is the probability of each class to be derived. N and M are two bounds used in the set of possible multiplicities. Lastly it must be given whether the model must be generally instanticiable or uninstanticiable.

In the procedure of generation of a random model firstly C classes are generated. Each class is abstract with the probability CA. For each generated class then optionally a parent class is assigned. Each class have a parent class with the probability CD. The parent is chosen randomly from all generated classes. It can not be the original class or a class that is directly or incrementally derived from the original class. Then for each pair of classes, also for each class with itself and for pairs with inverted elements, a binary association is generated with the probability A. The two multiplicities for the association are each selected randomly from the set of multiplicities containing $[0,1]$, $[1,1]$, [0,*], [1,*], $[0,N]$, $[N,*]$ and $[N,M]$.

Parameter	Value for instantiable models	Value for uninstantiable models
As	$\{2,5\}$	$\{15,20\}$
Cs	$\{10,25\}$	$\{50,100\}$
CAs	$\{5,10,20\}$	$\{5,10,20\}$
CDs	$\{5,25,50\}$	$\{5,25,50\}$
Ns	$\{2\}$	$\{2\}$
Ms	$\{10\}$	$\{10\}$
number of models	$\{5\}$	$\{5\}$

TABLE 4.1: Applied parameter values for generation of sets of random models

For the generation of sets of random models a set of values for each of this parameters is used as parameter, namely As, Cs, CAs, CDs, Ns, Ms. Additionally the number of models for each combination of the parameters values must be specified. It must also be specified whether all models must be generally instanticiable or uninstanticiable. The values from table 4.1 for the parameters for the generation of sets of random models are used for generally instanticiable and uninstanticiable models. For each of the two types of models 180 models are generated. Since other model aspects are not modeled, uninstantiability can only be achieved with conflicting multiplicities in association. This may be enhanced with the dependency hierarchy and the property of classes to be abstract. The presence of these instantiability problems are more probable with more classes. In the process of generating random model, also models that do not fulfil the selected property, of being generally instantiable or uninstantiable, were generated but discarded. With inappropriate parameters, that result in lots of generated models to be discarded, a lot of time may be needed. For this reason, generally instantiable models are searched with less classes and lower probabilities of each pair of classes to be associated. Uninstantiable models are searched with more classes and higher probabilities of each pair of classes to be associated.

The consistency check use case is applicated for the evaluation. It checks whether at least one object diagram can be generated for a class diagram in context of the search space represented by an instance finder configuration. Four parameters are used for the evaluation procedure. Firstly it is the set of models. Then Bs, a set of bounds B, must be given, which is used for the general bounds of simple configurations and the initial bounds for the clever generation procedure. Since there may be other influences on the time expenditures on the performing system, the number of runs per combination of model and B must be specified. Data generated on multiple

runs can be compared and outliers can be identified. The averaged time expenditures are used instead of single ones. Lastly, a maximum time must be specified. It is used for each consistency check.

Parameter	**Value for instantiable models**	**Value for uninstantiable models**
Bs	$\{[1,10],[1,100],[1,250]\}$	$\{[1,10],[1,100]\}$
number of runs	3	3
maximum time	180 seconds	60 seconds

TABLE 4.2: Applied parameter values for evaluation

The two sets of models are used for the generation. Table 4.2 shows the values for the other parameters for the evaluation procedure. Three processes are included in each run for a model. These are explained in the following. Ignoring other time consuming performances in the evaluation procedure, the overall needed time for the procedure is therefor limited to 874800 seconds (243 hours) for generally instantiable models and to 194400 seconds (54 hours) for uninstantiable models. Since the parameters are selected with the intend to not only show cases where the maximum time would be exceeded, the actual overall time expenditures of the two evaluation processes are much lower than these limits. For uninstantiable models the wider bounds $[1,250]$ are not contained in the parameters. Here, a more less maximum time of 60 seconds was chosen, because with the bounds $[1,100]$ already the maximum time is exceeded oftenly. Therefor, also no valuable time expenditures would be observed with wider bounds.

The first step in the overall evaluation procedure is to document the time expenditures. Further information is derived on this basis. A data entry is generated for each model. It documents the model generation parameter values and three time expenditures for each run in combination with each B. This contains the time needed for the clever generation procedure and the two time expenditures of performing the consistency check with a simple configuration and with the clever generated configuration. The latter one is -1 if the clever generation procedure fails in all runs for a model. The clever generation procedure either fails or does not fail in all runs for a model. Therefor, either all time expenditures for the consistency check on a model and clever generated configuration based on B are -1 or none of these are -1. This must be clarified since averaging sets of time expenditures containing -1 and accurate values must be avoided.

Information can be derived from the generated data in the evaluation procedure. For each B for each of the three kinds of time expenditures in each data entry the averaged time expenditure is computed. For uninstantiable models this must be always -1 for all consistency checks using the cleverly generated configurations. For generally instantiable models this can be -1 for some consistency checks using the cleverly generated configurations. If no configuration can be cleverly generated based on B for a generally instantiable model, the combination of model and B is ignored. In that case, this special information must be documented also. This must be done, because averaging sets of time expenditures containing -1 and accurate values must be avoided. For all applicated parameter values for the random model generation, e.g. for each 5 (minus the ignored) models, for each B for each of the three kinds of time expenditures the averaged time expenditure is computed. The

resulting data entries contains the model generation parameter values, the number of models whose results are averaged and three averaged time expenditures for each B. Then for each entry for each B the sum of the time expenditures of the clever generation procedure and the consistency check procedure with the resulting configuration is added. 0 instead of -1 is used for the computation of the sum when the consistency check procedure time expenditure is -1. The quotient of the averaged time expenditures of the consistency check with the simple configurations and the sum is added. This represents the speedup of the time expenditures using the feature against using the original functionality with not optimised configurations.

Several other parameters may influence the time expenditures. This contains the specs of the executing system or, to be more precise, the allocated resources. The evaluation is executed on 64-bit OS with a quad-core CPU with 2,5 GHz and 16GB RAM. USE version 5.1.0 and Java version 1.8 is used. Each execution of USE is done while using the JVM options `-Xms2G -Xmx2G`. For the model validator plugin the default settings are used.

Header	Meaning
A	Probability of each pair of classes to be part of binary associations, given in percent.
C	Number of classes.
CA	Probability of each class to be abstract, given in percent.
CD	Probability of each class to be derived, given in percent.
N	First bound for possible multiplicities.
M	Second bound for possible multiplicities.
...Ts ...	This is given for each of the next three headers. It represents the numbers of timeouts in the processes there.
VD ...	Averaged time expenditures for the consistency check process with the configurations with the following general bounds.
GC ...	Averaged time expenditures for the clever generation process with the following initial bounds.
VC ...	Averaged time expenditures for the consistency check process with the clever generated configurations based on the following initial bounds. The value is -1 if no configuration could be created.
Speedup ...	The speedup resulting from dividing the corresponding "VD" value by the sum of the "GC" and "VC" values. If the "VC" value is -1, then 0 is used instead. The speedup has suffix $+$ if there are timeouts only for corresponding "VD" value. The speedup has suffix $-$ if there are timeouts only for corresponding "GC" and/or "VC" values. The speedup has suffix ? if there are timeouts only for corresponding "GC" and/or "VC" values and also for the "VD" value.

TABLE 4.3: Meanings of headers in tables presenting evaluation data

Table 4.3 shows the meaning of headers in the following tables that present evaluation data. Regarding the speedup, there are three possible suffixes added to the speedup value. This depends on the presence of timeouts. For the interpretation of data the presence of the suffix ? is hindering. This means there were timeouts exceeded in both kind of processes that are compared. Timeouts exceeded with usage of a simple configuration and with generating and using a clever configuration. One may not now if the real time expenditure is less or more.

A	C	CA	CD	N	M	VD [1,10]	VD_Ts [1,10]	VD [1,100]	VD_Ts [1,100]	VD [1,250]	VD_Ts [1,250]	GC [1,10]	GC_Ts [1,10]	GC [1,100]	GC_Ts [1,100]	GC [1,250]	GC_Ts [1,250]	VC [1,10]	VC_Ts [1,10]	VC [1,100]	VC_Ts [1,100]	VC [1,250]	VC_Ts [1,250]	Speedup [1,10]	Speedup [1,100]	Speedup [1,250]
2	10	10	25	2	10	1,62	0	40,59	3	82,22	6	1,44	0	1,33	0	1,34	0	1,07	0	1,08	0	0,97	0	0,65	16,81+	35,69+
2	10	10	50	2	10	2,19	0	31,54	0	109,13	9	1,96	0	1,71	0	1,95	0	1,58	0	1,47	0	1,59	0	0,62	9,90	30,81+
2	10	10	5	2	10	1,44	0	3,98	0	23,90	0	1,44	0	1,13	0	1,42	0	0,95	0	0,73	0	0,98	0	0,60	2,14	9,97
2	10	5	25	2	10	1,43	0	89,31	0	141,36	9	1,09	0	1,20	0	1,57	0	0,71	0	1,10	0	0,95	0	0,79	38,79	56,02+
2	10	5	50	2	10	1,60	0	18,00	0	109,19	9	1,56	0	1,36	0	1,93	0	0,94	0	1,20	0	1,71	0	0,64	7,02	30,00+
2	10	5	5	2	10	1,82	0	74,60	6	106,10	6	1,57	0	1,45	0	1,51	0	1,07	0	1,08	0	1,37	0	0,69	29,40+	36,76+
2	10	20	25	2	10	1,08	0	2,97	0	43,67	1	0,97	0	0,97	0	0,98	0	0,65	0	0,64	0	0,66	0	0,67	1,84	26,54+
2	10	20	50	2	10	1,52	0	41,75	0	85,49	3	1,35	0	1,35	0	1,34	0	1,02	0	0,90	0	1,16	0	0,64	18,60	34,21+
2	10	20	5	2	10	1,11	0	48,66	3	87,62	6	0,96	0	0,98	0	1,12	0	0,65	0	0,65	0	0,79	0	0,69	29,72+	45,89+
2	25	5	25	2	10	1,93	0	93,90	4	173,90	11	1,11	0	1,45	0	1,96	0	0,65	0	1,28	0	1,30	0	1,10	34,33+	53,45+
2	25	5	50	2	10	7,81	0	112,68	9	165,47	12	1,58	0	1,72	0	1,72	0	1,28	0	1,21	0	1,30	0	2,74	38,37+	54,86+
2	25	5	5	2	10	1,87	0	38,70	0	119,71	6	1,56	0	1,31	0	1,56	0	1,27	0	1,03	0	0,95	0	0,66	16,50	47,61+
2	25	10	25	2	10	1,47	0	79,41	6	180,00	15	1,20	0	1,71	0	1,34	0	0,66	0	1,34	0	1,05	0	0,79	26,03+	75,28+
2	25	10	50	2	10	4,56	0	125,11	3	180,00	15	1,71	0	1,59	0	1,72	0	1,40	0	0,91	0	1,43	0	1,47	50,10+	57,02+
2	25	10	5	2	10	1,13	0	5,69	0	60,47	0	1,11	0	1,12	0	1,24	0	0,66	0	0,67	0	0,92	0	0,64	3,19	28,06
2	25	20	25	2	10	2,27	0	38,96	1	103,45	6	1,09	0	1,46	0	1,58	0	0,66	0	1,28	0	0,93	0	1,30	14,19+	41,27+
2	25	20	50	2	10	1,98	0	25,99	0	162,73	12	1,75	0	1,64	0	1,65	0	1,60	0	1,06	0	1,55	0	0,59	9,61	50,75+
2	25	20	5	2	10	1,20	0	3,99	0	35,96	0	1,14	0	1,44	0	1,20	0	0,90	0	0,89	0	0,91	0	0,59	1,71	17,05
5	10	10	25	2	10	1,68	0	19,00	0	112,82	8	1,75	0	1,50	0	1,62	0	0,90	0	1,53	0	1,42	0	0,64	6,27	37,14+
5	10	10	50	2	10	14,09	0	58,31	3	80,66	6	1,78	0	1,63	0	1,77	0	1,39	0	1,14	0	1,72	0	4,44	20,99+	23,17+
5	10	10	5	2	10	2,06	0	8,06	0	57,76	3	1,98	0	1,61	0	1,61	0	1,76	0	1,14	0	1,03	0	0,55	2,93	26,65+
5	10	5	25	2	10	1,41	0	93,10	3	168,09	9	1,12	0	1,75	0	1,61	0	0,77	0	1,41	0	1,03	0	0,75	29,46+	63,71+
5	10	5	50	2	10	37,94	3	88,23	5	147,72	9	1,27	0	1,12	0	1,50	0	0,78	0	0,93	0	1,03	0	18,45+	42,99+	58,32+

TABLE 4.4: Evaluation data for generally instantiable models (Part 1) The models are generated with the parameters from table 4.1. The evaluation is processed with parameters from table 4.2. See table 4.3 for meanings of headers. All time expenditures are given in seconds.

A	C	CA	CD	N	M	VD [1,10]	VD_Ts [1,10]	VD [1,100]	VD_Ts [1,100]	VD [1,250]	VD_Ts [1,250]	GC [1,10]	GC_Ts [1,10]	GC [1,100]	GC_Ts [1,100]	GC [1,250]	GC_Ts [1,250]	VC [1,10]	VC_Ts [1,10]	VC [1,100]	VC_Ts [1,100]	VC [1,250]	VC_Ts [1,250]	Speedup [1,10]	Speedup [1,100]	Speedup [1,250]
5	10	5	5	2	10	1,60	0	44,43	3	109,20	3	1,24	0	1,25	0	1,25	0	1,02	0	0,89	0	0,78	0	0,71	20,77+	53,97+
5	10	20	25	2	10	1,80	0	12,71	0	76,23	6	1,69	0	1,93	0	2,07	0	1,33	0	1,72	0	1,60	0	0,59	3,48	20,78+
5	10	20	50	2	10	1,54	0	69,37	3	122,99	6	1,32	0	1,58	0	1,34	0	0,88	0	1,53	0	0,72	0	0,70	22,31+	59,81+
5	10	20	5	2	10	1,70	0	2,63	0	13,28	0	1,54	0	1,55	0	2,09	0	1,23	0	1,08	0	1,77	0	0,61	1,00	3,44
5	25	5	25	2	10	38,19	3	112,04	7	180,00	15	1,75	0	1,41	0	1,58	0	1,21	0	0,98	0	1,03	0	12,92+	46,84+	68,89+
5	25	5	50	2	10	4,79	0	131,20	7	180,00	15	1,69	0	1,42	0	1,66	0	1,20	0	1,00	0	1,36	0	1,66	54,14+	59,54+
5	25	5	5	2	10	1,79	0	33,78	0	180,00	15	1,61	0	1,37	0	1,37	0	1,09	0	0,73	0	0,85	0	0,66	16,12	81,01+
5	25	10	25	2	10	1,60	0	15,33	0	179,76	14	1,79	0	1,50	0	1,97	0	1,05	0	1,17	0	1,54	0	0,57	5,75	51,21+
5	25	10	50	2	10	2,01	0	81,07	3	180,00	15	1,90	0	1,95	0	1,83	0	1,43	0	1,31	0	1,63	0	0,60	24,90+	52,01+
5	25	10	5	2	10	1,43	0	11,94	0	180,00	15	1,42	0	1,23	0	1,60	0	0,91	0	0,78	0	1,05	0	0,62	5,95	67,99+
5	25	20	25	2	10	1,20	0	15,08	0	164,71	12	1,24	0	1,59	0	2,06	0	0,66	0	1,28	0	1,24	0	0,63	5,26	49,83+
5	25	20	50	2	10	1,81	0	79,83	3	180,00	15	1,31	0	1,71	0	1,87	0	0,67	0	1,68	0	1,39	0	0,91	23,54+	55,19+
5	25	20	5	2	10	1,16	0	8,16	0	132,15	8	1,21	0	1,20	0	1,45	0	0,66	0	0,80	0	0,96	0	0,62	4,07	54,77+

TABLE 4.5: Evaluation data for generally instantiable models (Part 2) The models are generated with the parameters from table 4.1. The evaluation is processed with parameters from table 4.2. See table 4.3 for meanings of headers. All time expenditures are given in seconds.

A	C	CA	CD	N	M	B	Ignored models
2	25	5	50	2	10	$[1, 10]$	1
2	25	20	50	2	10	$[1, 10]$	1
5	25	5	25	2	10	$[1, 10]$	1
5	25	5	50	2	10	$[1, 10]$	2
5	25	10	50	2	10	$[1, 10]$	1
5	25	20	50	2	10	$[1, 10]$	1

TABLE 4.6: Numbers of models that are ignored in table 4.4 and table 4.5

Table 4.4 and table 4.5 show the time expenditures for the compared processes on generally instantiable models. There could have been failed clever generation processes, because the models must not be instantiable with every configuration. Table 4.6 shows that for seven models no configurations could be cleverly generated with $[1, 10]$. The combinations of $[1, 10]$ and these models were ignored in the averaging process. Not all averaged values are based on the same number of models. These seven cases are an example of how the consistency check result may be solely caused by the inappropriate too small search space. In all seven cases there can be found model instances with larger search space. All speedup values are useful. None of the speedup values has suffix ?. Also, the suffix $-$ is not present. Both of the compared processes are faster under specific circumstances. Consistency checks with simple configurations are often faster than with cleverly generating configurations, if the original represented search space is small. This becomes apparent with $[1, 10]$. Consistency checks with simple configurations are slower than with cleverly generating configurations, if the original represented search space is big. This becomes apparent with $[1, 100]$ and $[1, 250]$. For bigger search spaces the speedup may be even higher. The data shows some main results. Firstly, the time expenditures for the clever generation of configurations does not seem to be dependent on the original search space. They also does seem to be nearly constant. Secondly, the time expenditures for the consistency check with cleverly generated configurations does also seem to be nearly constant. Thirdly, the larger the original search space is, the larger the time expenditure is for consistency checks with simple configurations. Lastly, the larger the original search space is, the more time is saved for consistency checks with preceding redundant search space removal.

A	C	CA	CD	N	M	VD [1,10]	VD_Ts [1,10]	VD [1,100]	VD_Ts [1,100]	GC [1,10]	GC_Ts [1,10]	GC [1,100]	GC_Ts [1,100]	VC [1,10]	VC_Ts [1,10]	VC [1,100]	VC_Ts [1,100]	Speedup [1,10]	Speedup [1,100]
15	50	10	25	2	10	3,53	0	9,67	0	3,03	0	3,20	0	0,00	0	0,00	0	1,16	3,02
15	50	10	50	2	10	4,23	0	8,45	0	3,13	0	3,25	0	0,00	0	0,00	0	1,35	2,60
15	50	10	5	2	10	2,83	0	8,76	0	3,15	0	3,29	0	0,00	0	0,00	0	0,90	2,66
15	50	5	25	2	10	5,01	0	16,11	2	3,31	0	3,91	0	0,00	0	0,00	0	1,51	4,12+
15	50	5	50	2	10	7,36	0	39,76	9	4,66	0	4,75	0	0,00	0	0,00	0	1,58	8,37+
15	50	5	5	2	10	3,62	0	40,87	9	3,32	0	4,44	0	0,00	0	0,00	0	1,09	9,20+
15	100	5	25	2	10	6,73	0	60,00	15	9,41	0	9,82	0	0,00	0	0,00	0	0,72	6,11+
15	100	5	50	2	10	13,91	0	60,00	15	11,22	0	10,82	0	0,00	0	0,00	0	1,24	5,54+
15	100	5	5	2	10	5,63	0	60,00	15	10,48	0	10,63	0	0,00	0	0,00	0	0,54	5,64+
15	50	20	25	2	10	3,80	0	60,00	15	3,76	0	3,76	0	0,00	0	0,00	0	1,01	15,96+
15	50	20	50	2	10	3,99	0	60,00	15	3,21	0	3,10	0	0,00	0	0,00	0	1,24	19,33+
15	50	20	5	2	10	3,19	0	60,00	15	3,91	0	3,32	0	0,00	0	0,00	0	0,82	18,05+
15	100	10	25	2	10	6,42	0	60,00	15	10,98	0	11,36	0	0,00	0	0,00	0	0,59	5,28+
15	100	10	50	2	10	12,88	0	60,00	15	11,30	0	11,21	0	0,00	0	0,00	0	1,14	5,35+
15	100	10	5	2	10	5,16	0	60,00	15	10,70	0	10,18	0	0,00	0	0,00	0	0,48	5,90+
15	100	20	25	2	10	5,94	0	60,00	15	10,60	0	10,68	0	0,00	0	0,00	0	0,56	5,62+
15	100	20	50	2	10	9,28	0	60,00	15	10,54	0	10,80	0	0,00	0	0,00	0	0,88	5,56+
15	100	20	5	2	10	4,59	0	60,00	15	10,74	0	10,63	0	0,00	0	0,00	0	0,43	5,65+
20	50	10	25	2	10	9,12	0	60,00	15	3,76	0	3,87	0	0,00	0	0,00	0	2,42	15,51+
20	50	10	50	2	10	6,21	0	60,00	15	4,52	0	4,03	0	0,00	0	0,00	0	1,37	14,87+
20	50	10	5	2	10	2,77	0	60,00	15	3,66	0	4,08	0	0,00	0	0,00	0	0,76	14,70+
20	50	5	25	2	10	4,49	0	60,00	15	4,39	0	4,12	0	0,00	0	0,00	0	1,02	14,57+
20	50	5	50	2	10	20,65	3	60,00	15	3,42	0	4,17	0	0,00	0	0,00	0	6,03+	14,39+

TABLE 4.7: Evaluation data for uninstantiable models (Part 1) The models are generated with the parameters from table 4.1. The evaluation is processed with parameters from table 4.2. See table 4.3 for meanings of headers. All time expenditures are given in seconds.

A	C	CA	CD	N	M	VD [1,10]	VD_Ts [1,10]	VD [1,100]	VD_Ts [1,100]	GC [1,10]	GC_Ts [1,10]	GC [1,100]	GC_Ts [1,100]	VC [1,10]	VC_Ts [1,10]	VC [1,100]	VC_Ts [1,100]	Speedup [1,10]	Speedup [1,100]
20	50	5	5	2	10	2,47	0	60,00	15	3,29	0	4,34	0	0,00	0	0,00	0	0,75	13,82+
20	100	5	25	2	10	8,21	0	60,00	15	14,19	0	15,08	0	0,00	0	0,00	0	0,58	3,98+
20	100	5	50	2	10	24,52	0	60,00	15	15,07	0	15,61	0	0,00	0	0,00	0	1,63	3,84+
20	100	5	5	2	10	6,79	0	60,00	15	15,22	0	15,06	0	0,00	0	0,00	0	0,45	3,98+
20	50	20	25	2	10	2,85	0	60,00	15	3,53	0	3,48	0	0,00	0	0,00	0	0,81	17,24+
20	50	20	50	2	10	5,09	0	60,00	15	3,78	0	4,30	0	0,00	0	0,00	0	1,34	13,96+
20	50	20	5	2	10	3,30	0	60,00	15	3,96	0	3,90	0	0,00	0	0,00	0	0,83	15,38+
20	100	10	25	2	10	7,77	0	60,00	15	14,63	0	14,93	0	0,00	0	0,00	0	0,53	4,02+
20	100	10	50	2	10	14,81	0	60,00	15	15,16	0	15,91	0	0,00	0	0,00	0	0,98	3,77+
20	100	10	5	2	10	7,03	0	60,00	15	16,36	0	15,84	0	0,00	0	0,00	0	0,43	3,79+
20	100	20	25	2	10	6,79	0	60,00	15	15,23	0	14,82	0	0,00	0	0,00	0	0,45	4,05+
20	100	20	50	2	10	12,99	0	60,00	15	14,75	0	14,92	0	0,00	0	0,00	0	0,88	4,02+
20	100	20	5	2	10	5,69	0	60,00	15	14,79	0	14,79	0	0,00	0	0,00	0	0,38	4,06+

TABLE 4.8: Evaluation data for uninstantiable models (Part 2) The models are generated with the parameters from table 4.1. The evaluation is processed with parameters from table 4.2. See table 4.3 for meanings of headers. All time expenditures are given in seconds.

Table 4.7 and table 4.8 show the time expenditures for the compared processes on uninstantiable models. Since those are uninstantiable, no configurations should be generated with the clever generation procedure. The time expenditures for consistency checks with cleverly generated configurations must always be -1. Since 0 is used instead of -1 in the averaging procedure, all averaged values therefor must be 0. Here, all averaged values for "VC $[1,10]$" and "VC $[1,100]$" are 0. All speedup values are useful. None of the speedup values has suffix ?. Also, the suffix $-$ is not present. Both of the compared processes are faster under specific circumstances. Consistency checks with simple configurations are often faster than with cleverly generating configurations if the original represented search space is small. This becomes apparent with $[1,10]$. Consistency checks with simple configurations are slower than with cleverly generated configurations if the original represented

search space is big. This becomes apparent with $[1, 100]$. The data shows some main results. Firstly, the larger the original search space is, the larger the time expenditure is for consistency checks with simple configurations. Secondly, the time expenditures for the clever generation of configurations does not seem to be dependent on the original search space. Lastly, the larger the original search space is, the more time is saved for consistency checks with preceding redundant search space removal.

Comparing the time expenditures for generally instantiable and uninstantiable models, the data from table 4.4, 4.5, 4.7 and 4.8 shows also some other results. While the time expenditures for the clever generation of configurations does seem to be nearly constant for generally instantiable models, they do not seem to be constant for uninstantiable models. They are also lower for generally instantiable models.

The data has shown that the larger the original search space is, the more time is saved for consistency checks with preceding redundant search space removal. Therefor, the extended functionality brings benefits. Since the absolute additional time needed for small search spaces is very low, it can be ignored that more time is needed for small search spaces with the extended functionality.

The evaluation of the comparisons and validation features is associated with quite a lot of effort. This also applies to the evaluation of some aspects of the clever generation feature. The opinions of experts or users on the usability of this features could be collected and analysed. The number of such persons is not large. Also, the communication and data collection would be very time consuming. Another idea on evaluation is to determine the influence of the usage of each feature on validation and verification processes. Regarding the comparisons and validation features, data of real projects is needed here. This implies that the two features are already used in real projects. One idea is to compare time expenditure of projects. Each with and without the usage of the feature. Running projects twice underlies crucial influences that may prevent deriving useful information of collected data. In the paradigm that each two runs are done by different people, this fact alone may cause different time expenditures. In the paradigm that each two runs are done by the same people, one may need less time on the second run because of knowledge generated in the first run. Again, this fact alone may cause different time expenditures.

Chapter 5

Conclusion and outlook

Aspects of the USE instance finder plugin, that could be improved, were presented. Extended functionality is implemented. This includes the missing possibility to compare configurations automised on the first hand. On the other hand, this also includes the missing possibility to create new configurations, so that at least one instance can be found with each of them. Lastly, the possibility to find specified invalid configuration aspects was missing. To avert these three problems, the functionality of the USE instance finder has been extended. Now configurations can be compared automised. In addition, new configurations can be created in a very flexible way. Finally, some validity rules for configurations have been established. For configurations, it is now possible to check if they are valid with respect to the rules. For configurations that are not valid, applicable corrections are suggested. This supports the process of identifying aspects of configurations that cause invalidity and to correct them.

Regarding time saving in model V&V, it was evaluated if at least the extended possibility to generate new configurations lowers time expenditures. The evaluation was performed under representative circumstances. Many scenarios were evaluated in a wide variety of several dimensions. The result can be divided into two cases. Similar or negligibly more time is required with the extended functionality in the scenarios where little time was already required with original functionality. This can be ignored since the absolute additional time is very low. Much less time is required with the extended functionality in the scenarios in which a lot of time was already required with original functionality. This is crucial because the absolute saved time can be very high.

In the evaluation only an extension of the USE instance finder has been respected. However, it is obvious that the other extensions also support model V&V processes. Regarding the validity of instance finder configurations, there are concepts implemented and presented. They support the presentation of this complex kind of information and provide much derived information. Regarding the suitability of configurations, the new generation feature and also the validity feature support the removal of unsuitable search space. This results in lower time expenditures in model V&V. Therefor, the research hypothesis holds. Since it holds, there is a communicable way of ensuring validity and suitability of bounds specifications for the USE instance finder of UML class diagrams. This is also the answer to the research question. There may be other problems with the instance finder and also other possibilities to solve such problems, while the answer to the research question remains the same.

The objectives of this work have been achieved. But there a several other points of further work. Selected points of further work are outlined in the following.

The developed concept for comparison of configurations does not include all types of aspects of configurations. For several aspects the comparison is not actually processed. Instead, it is worked with the result that these aspects are ignored

in comparison. The implementation already provides the structure for aspect specific comparison of all aspects. The implementations that compute those constant results only need to be simply replaced by appropriate implementations. But before, the aspect specific comparison concepts must be developed. Since there are other crucial problems to be dealt with, here only selected aspects are respected with the implemented comparison concept.

The developed concept for validation of configurations could be also applied in more complex procedures. It is always possible to calculate the resulting configuration for each proposed correction. This can be done iteratively also. Therefor, there may exist also chains of corrections for an invalid configuration. It can be determined which chains of corrections are the shortest. A system for automised application of corrections can be implemented. An approach may be to additionally classify corrections to handling them differently. Sets of valid configurations resulting from different chaining of the application of corrections, could be used as suggested corrections of invalid configurations. An automatised selection is possible on this basis. The chain of corrections to be applied can also be used with the comparison functionality to convey the difference between an original invalid configuration and its valid corrected configuration. The concept model checking can also be applied here. The absence of finite chains of corrections whose application would result in a valid configuration can be proved. There can be infinite chains of corrections too. However, those never lead to a valid configuration. It can be pointed out in advance that the application of validation and correction cannot produce this desired result. Instead, a configuration would have to be changed in some other way than by the suggested corrections. Also, for any other change of the configuration it could be calculated in advance if the resulting configuration is valid. In case it is not, it could be calculated again if there is at least one valid corrected configuration which can be produced by a chain of corrections from the validation functionality. Whether a specific modification helps the configuration to become valid could be computed for each modification of an invalid configuration.

The concept of CSPs is used for redundant search space removal on search space represented by USE instance finder configuration. The implemented extended functionality does not respect the OCL constraints of the UML/OCL models when building a CSP for such a model for redundant search space removal. Clarisó, González, and Cabot (2019) already presented approaches to derive CSP constraints for the OCL constraints. Beside the currently respected elements class and associations and the aspects of classes to be abstract or derived, also the elements attribute, types and constraints become involved. With this, more redundant search space may be removed or identified. Since the whole procedure becomes more complex, also additional time expenditures may are implied.

Bibliography

Biere, Armin and Daniel Kröning (2018). "SAT-Based Model Checking". In: *Handbook of Model Checking*. Ed. by Edmund M. Clarke, Thomas A. Henzinger, and Helmut Veith. Cham: Springer International Publishing. ISBN: 9783319105758. DOI: 10.1007/978-3-319-10575-8_1.

Brambilla, Marco, Jordi Cabot, and Manuel Wimmer (2012). *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers. ISBN: 9781608458820. DOI: 10.2200/S00441ED1V01Y201208SWE001.

Cabot, Jordi, Robert Clarisó, and Daniel Riera (Mar. 2014). "On the verification of UML/OCL class diagrams using constraint programming". In: *Journal of Systems and Software* 93, pp. 1–23. DOI: 10.1016/j.jss.2014.03.023.

Clarisó, Robert, Carlos A. González, and Jordi Cabot (Apr. 2019). "Smart Bound Selection for the Verification of UML/OCL Class Diagrams". In: vol. 45. 4, pp. 412–426. DOI: 10.1109/TSE.2017.2777830.

Clarke, Edmund M., Thomas A. Henzinger, and Helmut Veith (2018). "Introduction to Model Checking". In: *Handbook of Model Checking*. Ed. by Edmund M. Clarke, Thomas A. Henzinger, and Helmut Veith. Cham: Springer International Publishing. ISBN: 9783319105758. DOI: 10.1007/978-3-319-10575-8_1.

Cooper, Martin C., Wafa Jguirim, and David A. Cohen (2018). "Domain Reduction for Valued Constraints by Generalising Methods from CSP". In: *Principles and Practice of Constraint Programming*. Ed. by John Hooker. Cham: Springer International Publishing, pp. 64–80. ISBN: 9783319983349.

Florez, Hector and Marcelo Leon (2018). "Model Driven Engineering Approach to Configure Software Reusable Components". In: *Applied Informatics*. Ed. by Hector Florez, Cesar Diaz, and Jaime Chavarriaga. Cham: Springer International Publishing, pp. 352–363. ISBN: 9783030015350.

Ghedira, Khaled (2013). *Constraint Satisfaction Problems: CSP Formalisms and Techniques*. 1. Aufl. Computer engineering and IT series. Cham: Wiley-ISTE. ISBN: 184821460X and 1118574575 and 9781118574577 and 9781118574577.

Giraldo, Fáber D. et al. (June 2018). "Considerations about quality in model-driven engineering". In: *Software Quality Journal* 26.2, pp. 685–750. DOI: 10.1007/s11219-016-9350-6.

Gogolla, Martin, Frank Hilken, and Khanh-Hoang Doan (2018). "Achieving Model Quality through Model Validation, Verification and Exploration". In: *Computer Languages, Systems & Structures* 54, pp. 474–511. DOI: 10.1016/j.cl.2017.10.001.

"IEEE Standard Glossary of Software Engineering Terminology" (Dec. 1990). In: *IEEE Std 610.12-1990*. DOI: 10.1109/IEEESTD.1990.101064.

ISO/IEC 19501:2005 (Apr. 2005). https://www.iso.org/standard/32620.html. Abruf: 06.12.2019.

Jácome, Santiago, Juan Ferreira, and Maria Corral-Diaz (Oct. 2017). "Software Development Tools in Model-Driven Engineering". In: DOI: 10.1109/CONISOFT.2017.00024.

Ludewig, Jochen (2000). "Software Engineering in the Year 2000 Minus and Plus Ten". In: *IEEE Std 610.12-1990*.

Ludewig, Jochen and Horst Lichter (2013). *Software Engineering. Grundlagen, Menschen, Prozesse, Techniken*. 3., korrigierte Aufl. Heidelberg: dpunkt.verlag. ISBN: 9783864912986.

Muller, Pierre-Alain et al. (July 2012). "Modeling modeling modeling". In: *Software & Systems Modeling* 11.3, pp. 347–359. DOI: 10.1007/s10270-010-0172-x.

Object Constraint Language 2.4 (Feb. 2014). https://www.omg.org/spec/OCL/2.4/PDF. Abruf: 11.12.2019.

Przigoda, Nils, Judith Wille Robert and Przigoda, and Rolf Drechsler (2018). *Automated Validation and Verification of UML/OCL Models Using Satisfiability Solvers*. Cham: Springer. ISBN: 9783319728148.

Selic, Bran (2006). "UML 2: A model-driven development tool". In: *IBM Systems Journal* 45.3, pp. 607–620.

Seshia, Sanjit A., Natasha Sharygina, and Stavros Tripakis (2018). "Modeling for Verification". In: *Handbook of Model Checking*. Ed. by Edmund M. Clarke, Thomas A. Henzinger, and Helmut Veith. Cham: Springer International Publishing. ISBN: 9783319105758. DOI: 10.1007/978-3-319-10575-8_1.

Unified Modeling Language (UML), Version 2.5.1 (Dec. 2017). https://www.omg.org/spec/UML/2.5.1/PDF. Abruf: 16.07.2019.

USE documentation (Mar. 2007). http://www.db.informatik.uni-bremen.de/projects/use/use-documentation.pdf. Abruf: 10.01.2020.

USE: UML-based Specification Environment (2020). https://sourceforge.net/projects/useocl/. Abruf: 10.01.2020.

Whittle, Jon, John Hutchinson, and Mark Rouncefield (Mar. 2014). "The State of Practice in Model-Driven Engineering". In: *IEEE Software* 31.3, pp. 79–85. DOI: 10.1109/MS.2013.65.

Appendix A

Example output for comparing configurations

```
1  1. config1 <= config2 :
2  Comparisons that are ignored
3     Person_lName is ignored.
4     Pet is ignored.
5     Parenthood is ignored.
6     Person_id is ignored.
7     Person_fName is ignored.
8     Individual_id is ignored.
9     Person_nicknames is ignored.
10    Pet_nickName is ignored.
11    forbiddensharing is ignored.
12    Person is ignored.
13    aggregationcyclefreeness is ignored.
14    Individual is ignored.
15    Person_favoriteNumber is ignored.
16    Integer is ignored.
17    Pet_id is ignored.
18    Person_yearB is ignored.
19    PetSitting is ignored.
20    String is ignored.
21 Comparisons that classify as equal
22    String enabled value is both false.
23    Invariant settings are empty.
24    Person_min and Person_max classify equality ( left: [ Person_min=1 ;
          Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
25    Parenthood_min and Parenthood_max classify equality ( left: [
          Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
          =1 ; Parenthood_max=1 ] ).
26    PetSitting_min and PetSitting_max classify equality ( left: [
          PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
          =1 ; PetSitting_max=1 ] ).
27    Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
          =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
28    Real enabled value is both false.
29    Individual_min and Individual_max classify equality ( left: [
          Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
          =1 ; Individual_max=1 ] ).
30 Comparisons that classify as right is broad
31    Integer_min and Integer_max classify as right is broad ( left: [
          Integer_min=-10 ; Integer_max=10 ] ; right: [ Integer_min=-100 ;
           Integer_max=100 ] ).
32
33 2. config1 >= config3 :
34 Comparisons that are ignored
35    String is ignored.
36    Person_fName is ignored.
```

```
37      Person is ignored.
38      PetSitting is ignored.
39      Individual is ignored.
40      Pet is ignored.
41      Person_lName is ignored.
42      Parenthood is ignored.
43      Person_favoriteNumber is ignored.
44      Pet_id is ignored.
45      aggregationcyclefreeness is ignored.
46      forbiddensharing is ignored.
47      Integer is ignored.
48      Individual_id is ignored.
49      Pet_nickName is ignored.
50      Person_yearB is ignored.
51      Person_nicknames is ignored.
52      Person_id is ignored.
53    Comparisons that classify as equal
54      String enabled value is both false.
55      PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
56      Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
57      Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
58      Invariant settings are empty.
59      Real enabled value is both false.
60      Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
61      Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
62    Comparisons that classify as left is broad
63      Integer_min and Integer_max classify as left is broad ( left: [
            Integer_min=-10 ; Integer_max=10 ] ; right: [ Integer_min=0 ;
            Integer_max=10 ] ).
64
65    3. config1 != config4 :
66    Comparisons that are ignored
67      Person_favoriteNumber is ignored.
68      Person_nicknames is ignored.
69      Individual_id is ignored.
70      Pet is ignored.
71      Person_fName is ignored.
72      String is ignored.
73      Person_yearB is ignored.
74      Individual is ignored.
75      Integer is ignored.
76      PetSitting is ignored.
77      Pet_id is ignored.
78      aggregationcyclefreeness is ignored.
79      Person is ignored.
80      Person_lName is ignored.
81      Parenthood is ignored.
82      Person_id is ignored.
83      Pet_nickName is ignored.
84      forbiddensharing is ignored.
85    Comparisons that classify as equal
86      Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
87      Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
```

```
88     Parenthood_min and Parenthood_max classify equality ( left: [
           Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
           =1 ; Parenthood_max=1 ] ).
89     String enabled value is both false.
90     Real enabled value is both false.
91     Individual_min and Individual_max classify equality ( left: [
           Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
           =1 ; Individual_max=1 ] ).
92     PetSitting_min and PetSitting_max classify equality ( left: [
           PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
           =1 ; PetSitting_max=1 ] ).
93     Invariant settings are empty.
94   Comparisons that classify as left prelimary overlapping
95     Integer_min and Integer_max classify as left is overlapping ( left:
           [ Integer_min=-10 ; Integer_max=10 ] ; right: [ Integer_min=0 ;
           Integer_max=100 ] ).
96
97   4. config1 != config5 :
98   Comparisons that are ignored
99     Individual is ignored.
100    Person_yearB is ignored.
101    PetSitting is ignored.
102    Pet is ignored.
103    Person_lName is ignored.
104    Person is ignored.
105    Person_fName is ignored.
106    Pet_nickName is ignored.
107    Parenthood is ignored.
108    Person_id is ignored.
109    forbiddensharing is ignored.
110    String is ignored.
111    Person_favoriteNumber is ignored.
112    aggregationcyclefreeness is ignored.
113    Integer is ignored.
114    Pet_id is ignored.
115    Individual_id is ignored.
116    Person_nicknames is ignored.
117  Comparisons that classify as equal
118    Invariant settings are empty.
119    Parenthood_min and Parenthood_max classify equality ( left: [
           Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
           =1 ; Parenthood_max=1 ] ).
120    Real enabled value is both false.
121    PetSitting_min and PetSitting_max classify equality ( left: [
           PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
           =1 ; PetSitting_max=1 ] ).
122    Individual_min and Individual_max classify equality ( left: [
           Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
           =1 ; Individual_max=1 ] ).
123    Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
           =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
124    String enabled value is both false.
125    Person_min and Person_max classify equality ( left: [ Person_min=1 ;
            Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
126  Comparisons that classify as left prelimary overlapping
127    Integer_min and Integer_max classify as left is overlapping ( left:
           [ Integer_min=-10 ; Integer_max=10 ] ; right: [ Integer_min=0 ;
           Integer_max=1000 ] ).
128
129  5. config1 >= config6 :
130  Comparisons that are ignored
131    PetSitting is ignored.
132    forbiddensharing is ignored.
```

```
133     Person_favoriteNumber is ignored.
134     Pet is ignored.
135     Integer is ignored.
136     Pet_id is ignored.
137     Person_id is ignored.
138     Person is ignored.
139     Person_yearB is ignored.
140     Person_fName is ignored.
141     Person_lName is ignored.
142     Individual is ignored.
143     Pet_nickName is ignored.
144     Parenthood is ignored.
145     aggregationcyclefreeness is ignored.
146     String is ignored.
147     Person_nicknames is ignored.
148     Individual_id is ignored.
149  Comparisons that classify as equal
150     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
151     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
152     Real enabled value is both false.
153     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
154     String enabled value is both false.
155     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
156     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
157     Invariant settings are empty.
158  Comparisons that classify as left is broad
159     Integer_min and Integer_max classify as left is broad ( left: [
            Integer_min=-10 ; Integer_max=10 ] ; right: [ Integer_min=-10 ;
            Integer_max=0 ] ).
160
161  6. config1 != config7 :
162  Comparisons that are ignored
163     String is ignored.
164     forbiddensharing is ignored.
165     PetSitting is ignored.
166     Person is ignored.
167     Person_nicknames is ignored.
168     aggregationcyclefreeness is ignored.
169     Person_fName is ignored.
170     Pet_id is ignored.
171     Person_favoriteNumber is ignored.
172     Pet is ignored.
173     Individual is ignored.
174     Person_id is ignored.
175     Person_yearB is ignored.
176     Parenthood is ignored.
177     Integer is ignored.
178     Individual_id is ignored.
179     Person_lName is ignored.
180     Pet_nickName is ignored.
181  Comparisons that classify as equal
182     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
183     Invariant settings are empty.
```

```
184     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
            Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
185     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
186     Real enabled value is both false.
187     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
188     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
189     String enabled value is both false.
190   Comparisons that classify as left disjoint
191     Integer_min and Integer_max classify as left disjoint ( left: [
            Integer_min=-10 ; Integer_max=10 ] ; right: [ Integer_min=100 ;
            Integer_max=1000 ] ).
192
193   7. config1 != config8 :
194   Comparisons that are ignored
195     aggregationcyclefreeness is ignored.
196     Individual is ignored.
197     Person_id is ignored.
198     Person_fName is ignored.
199     Integer is ignored.
200     Person_lName is ignored.
201     Person_yearB is ignored.
202     Person_nicknames is ignored.
203     Pet_nickName is ignored.
204     PetSitting is ignored.
205     Pet is ignored.
206     String is ignored.
207     forbiddensharing is ignored.
208     Pet_id is ignored.
209     Individual_id is ignored.
210     Person is ignored.
211     Person_favoriteNumber is ignored.
212     Parenthood is ignored.
213   Comparisons that classify as equal
214     Real enabled value is both false.
215     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
216     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
217     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
218     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
219     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
            Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
220     String enabled value is both false.
221     Invariant settings are empty.
222   Comparisons that classify as right disjoint
223     Integer_min and Integer_max classify as right disjoint ( left: [
            Integer_min=-10 ; Integer_max=10 ] ; right: [ Integer_min
            =-100000 ; Integer_max=-10000 ] ).
224
225   8. config2 >= config3 :
226   Comparisons that are ignored
227     String is ignored.
```

```
228     Person_id is ignored.
229     Individual_id is ignored.
230     Integer is ignored.
231     Pet_id is ignored.
232     Individual is ignored.
233     forbiddensharing is ignored.
234     Person_favoriteNumber is ignored.
235     Parenthood is ignored.
236     aggregationcyclefreeness is ignored.
237     Pet_nickName is ignored.
238     Pet is ignored.
239     Person_nicknames is ignored.
240     Person is ignored.
241     Person_fName is ignored.
242     PetSitting is ignored.
243     Person_lName is ignored.
244     Person_yearB is ignored.
245 Comparisons that classify as equal
246     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
247     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
248     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
249     Invariant settings are empty.
250     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
251     String enabled value is both false.
252     Real enabled value is both false.
253     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
254 Comparisons that classify as left is broad
255     Integer_min and Integer_max classify as left is broad ( left: [
            Integer_min=-100 ; Integer_max=100 ] ; right: [ Integer_min=0 ;
            Integer_max=10 ] ).
256
257 9. config2 >= config4 :
258 Comparisons that are ignored
259     Person_yearB is ignored.
260     Pet is ignored.
261     Individual_id is ignored.
262     Person_lName is ignored.
263     Parenthood is ignored.
264     Individual is ignored.
265     aggregationcyclefreeness is ignored.
266     Pet_nickName is ignored.
267     String is ignored.
268     Person_id is ignored.
269     PetSitting is ignored.
270     forbiddensharing is ignored.
271     Person is ignored.
272     Person_fName is ignored.
273     Person_favoriteNumber is ignored.
274     Pet_id is ignored.
275     Integer is ignored.
276     Person_nicknames is ignored.
277 Comparisons that classify as equal
278     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
```

```
279     Real enabled value is both false.
280     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
281     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
282     Invariant settings are empty.
283     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
284     String enabled value is both false.
285     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
286   Comparisons that classify as left is broad
287     Integer_min and Integer_max classify as left is broad ( left: [
            Integer_min=-100 ; Integer_max=100 ] ; right: [ Integer_min=0 ;
            Integer_max=100 ] ).
288
289   10. config2 != config5 :
290   Comparisons that are ignored
291     String is ignored.
292     Person_fName is ignored.
293     Integer is ignored.
294     Pet_id is ignored.
295     forbiddensharing is ignored.
296     Individual is ignored.
297     Parenthood is ignored.
298     Person_lName is ignored.
299     Pet_nickName is ignored.
300     Person is ignored.
301     aggregationcyclefreeness is ignored.
302     PetSitting is ignored.
303     Person_nicknames is ignored.
304     Individual_id is ignored.
305     Pet is ignored.
306     Person_favoriteNumber is ignored.
307     Person_id is ignored.
308     Person_yearB is ignored.
309   Comparisons that classify as equal
310     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
311     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
312     Invariant settings are empty.
313     Real enabled value is both false.
314     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
315     String enabled value is both false.
316     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
317     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
318   Comparisons that classify as left prelimary overlapping
319     Integer_min and Integer_max classify as left is overlapping ( left:
            [ Integer_min=-100 ; Integer_max=100 ] ; right: [ Integer_min=0
            ; Integer_max=1000 ] ).
320
321   11. config2 >= config6 :
322   Comparisons that are ignored
323     Person_id is ignored.
```

```
324     Pet_id is ignored.
325     Individual_id is ignored.
326     PetSitting is ignored.
327     Pet is ignored.
328     Person_fName is ignored.
329     Integer is ignored.
330     forbiddensharing is ignored.
331     String is ignored.
332     Individual is ignored.
333     Person_nicknames is ignored.
334     Parenthood is ignored.
335     Person_favoriteNumber is ignored.
336     Person_yearB is ignored.
337     Person is ignored.
338     aggregationcyclefreeness is ignored.
339     Person_lName is ignored.
340     Pet_nickName is ignored.
341 Comparisons that classify as equal
342     Real enabled value is both false.
343     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
344     String enabled value is both false.
345     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
346     Invariant settings are empty.
347     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
348     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
349     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
350 Comparisons that classify as left is broad
351     Integer_min and Integer_max classify as left is broad ( left: [
            Integer_min=-100 ; Integer_max=100 ] ; right: [ Integer_min=-10
            ; Integer_max=0 ] ).
352
353 12. config2 != config7 :
354 Comparisons that are ignored
355     Individual_id is ignored.
356     Person is ignored.
357     Person_yearB is ignored.
358     aggregationcyclefreeness is ignored.
359     String is ignored.
360     Person_nicknames is ignored.
361     PetSitting is ignored.
362     Person_lName is ignored.
363     Integer is ignored.
364     Pet is ignored.
365     Person_favoriteNumber is ignored.
366     Pet_nickName is ignored.
367     forbiddensharing is ignored.
368     Person_fName is ignored.
369     Person_id is ignored.
370     Individual is ignored.
371     Pet_id is ignored.
372     Parenthood is ignored.
373 Comparisons that classify as equal
374     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
```

```
375     Real enabled value is both false.
376     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
377     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
378     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
379     String enabled value is both false.
380     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
381     Invariant settings are empty.
382   Comparisons that classify as left prelimary overlapping
383     Integer_min and Integer_max classify as left is overlapping ( left:
            [ Integer_min=-100 ; Integer_max=100 ] ; right: [ Integer_min
            =100 ; Integer_max=1000 ] ).
384
385   13. config2 != config8 :
386   Comparisons that are ignored
387     aggregationcyclefreeness is ignored.
388     Person_favoriteNumber is ignored.
389     Person_lName is ignored.
390     Pet_id is ignored.
391     String is ignored.
392     Person is ignored.
393     PetSitting is ignored.
394     Integer is ignored.
395     Individual is ignored.
396     Pet is ignored.
397     Parenthood is ignored.
398     forbiddensharing is ignored.
399     Pet_nickName is ignored.
400     Person_id is ignored.
401     Person_fName is ignored.
402     Individual_id is ignored.
403     Person_nicknames is ignored.
404     Person_yearB is ignored.
405   Comparisons that classify as equal
406     Real enabled value is both false.
407     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
408     Invariant settings are empty.
409     String enabled value is both false.
410     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
411     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
412     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
413     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
414   Comparisons that classify as right disjoint
415     Integer_min and Integer_max classify as right disjoint ( left: [
            Integer_min=-100 ; Integer_max=100 ] ; right: [ Integer_min
            =-100000 ; Integer_max=-10000 ] ).
416
417   14. config3 <= config4 :
418   Comparisons that are ignored
419     Integer is ignored.
```

```
420     Individual_id is ignored.
421     Person_lName is ignored.
422     Person_yearB is ignored.
423     Parenthood is ignored.
424     Pet is ignored.
425     Person_fName is ignored.
426     Person_favoriteNumber is ignored.
427     aggregationcyclefreeness is ignored.
428     Person is ignored.
429     Person_id is ignored.
430     forbiddensharing is ignored.
431     Pet_nickName is ignored.
432     Pet_id is ignored.
433     Individual is ignored.
434     PetSitting is ignored.
435     Person_nicknames is ignored.
436     String is ignored.
437  Comparisons that classify as equal
438     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
439     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
440     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
441     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
442     Invariant settings are empty.
443     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
444     Real enabled value is both false.
445     String enabled value is both false.
446  Comparisons that classify as right is broad
447     Integer_min and Integer_max classify as right is broad ( left: [
            Integer_min=0 ; Integer_max=10 ] ; right: [ Integer_min=0 ;
            Integer_max=100 ] ).
448
449  15. config3 <= config5 :
450  Comparisons that are ignored
451     Person_fName is ignored.
452     Integer is ignored.
453     Pet_nickName is ignored.
454     String is ignored.
455     Individual is ignored.
456     Person_favoriteNumber is ignored.
457     Pet is ignored.
458     Pet_id is ignored.
459     forbiddensharing is ignored.
460     aggregationcyclefreeness is ignored.
461     Parenthood is ignored.
462     Person_id is ignored.
463     Person is ignored.
464     Person_lName is ignored.
465     PetSitting is ignored.
466     Individual_id is ignored.
467     Person_nicknames is ignored.
468     Person_yearB is ignored.
469  Comparisons that classify as equal
470     String enabled value is both false.
471     Real enabled value is both false.
```

```
472     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
473     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
474     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
475     Invariant settings are empty.
476     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
477     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
478 Comparisons that classify as right is broad
479     Integer_min and Integer_max classify as right is broad ( left: [
            Integer_min=0 ; Integer_max=10 ] ; right: [ Integer_min=0 ;
            Integer_max=1000 ] ).
480
481 16. config3 != config6 :
482 Comparisons that are ignored
483     Person_nicknames is ignored.
484     Pet_id is ignored.
485     Parenthood is ignored.
486     forbiddensharing is ignored.
487     Individual is ignored.
488     Person_lName is ignored.
489     String is ignored.
490     Person_fName is ignored.
491     Person_yearB is ignored.
492     Pet is ignored.
493     PetSitting is ignored.
494     Person_favoriteNumber is ignored.
495     Integer is ignored.
496     Person_id is ignored.
497     Individual_id is ignored.
498     Person is ignored.
499     Pet_nickName is ignored.
500     aggregationcyclefreeness is ignored.
501 Comparisons that classify as equal
502     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
503     Real enabled value is both false.
504     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
505     Invariant settings are empty.
506     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
507     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
508     String enabled value is both false.
509     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
510 Comparisons that classify as right prelimary overlapping
511     Integer_min and Integer_max classify as right is overlapping ( left:
             [ Integer_min=0 ; Integer_max=10 ] ; right: [ Integer_min=-10 ;
             Integer_max=0 ] ).
512
513 17. config3 != config7 :
514 Comparisons that are ignored
```

```
515     Individual is ignored.
516     Person_id is ignored.
517     Individual_id is ignored.
518     Parenthood is ignored.
519     Person_nicknames is ignored.
520     PetSitting is ignored.
521     Person is ignored.
522     Person_lName is ignored.
523     Person_favoriteNumber is ignored.
524     Person_yearB is ignored.
525     Integer is ignored.
526     aggregationcyclefreeness is ignored.
527     Pet is ignored.
528     Pet_nickName is ignored.
529     Pet_id is ignored.
530     Person_fName is ignored.
531     String is ignored.
532     forbiddensharing is ignored.
533  Comparisons that classify as equal
534     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
535     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
536     String enabled value is both false.
537     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
538     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
539     Real enabled value is both false.
540     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
541     Invariant settings are empty.
542  Comparisons that classify as left disjoint
543     Integer_min and Integer_max classify as left disjoint ( left: [
            Integer_min=0 ; Integer_max=10 ] ; right: [ Integer_min=100 ;
            Integer_max=1000 ] ).
544
545  18. config3 != config8 :
546  Comparisons that are ignored
547     Individual is ignored.
548     Person_lName is ignored.
549     Person_id is ignored.
550     Integer is ignored.
551     PetSitting is ignored.
552     Parenthood is ignored.
553     Person_fName is ignored.
554     String is ignored.
555     Person is ignored.
556     Pet is ignored.
557     Person_favoriteNumber is ignored.
558     Person_nicknames is ignored.
559     Pet_id is ignored.
560     Pet_nickName is ignored.
561     aggregationcyclefreeness is ignored.
562     Individual_id is ignored.
563     forbiddensharing is ignored.
564     Person_yearB is ignored.
565  Comparisons that classify as equal
```

```
566     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
567     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
568     Invariant settings are empty.
569     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
570     Real enabled value is both false.
571     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
572     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
573     String enabled value is both false.
574 Comparisons that classify as right disjoint
575     Integer_min and Integer_max classify as right disjoint ( left: [
            Integer_min=0 ; Integer_max=10 ] ; right: [ Integer_min=-100000
            ; Integer_max=-10000 ] ).
576
577 19. config4 <= config5 :
578 Comparisons that are ignored
579     Person is ignored.
580     Person_nicknames is ignored.
581     forbiddensharing is ignored.
582     Integer is ignored.
583     Person_yearB is ignored.
584     PetSitting is ignored.
585     Person_fName is ignored.
586     Parenthood is ignored.
587     Individual_id is ignored.
588     Pet_nickName is ignored.
589     Person_favoriteNumber is ignored.
590     Person_lName is ignored.
591     Pet_id is ignored.
592     Pet is ignored.
593     String is ignored.
594     Individual is ignored.
595     aggregationcyclefreeness is ignored.
596     Person_id is ignored.
597 Comparisons that classify as equal
598     Real enabled value is both false.
599     Invariant settings are empty.
600     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
601     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
602     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
603     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
604     String enabled value is both false.
605     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
606 Comparisons that classify as right is broad
607     Integer_min and Integer_max classify as right is broad ( left: [
            Integer_min=0 ; Integer_max=100 ] ; right: [ Integer_min=0 ;
            Integer_max=1000 ] ).
608
```

```
609 20. config4 != config6 :
610 Comparisons that are ignored
611     PetSitting is ignored.
612     Person is ignored.
613     Person_id is ignored.
614     Person_yearB is ignored.
615     Pet is ignored.
616     Person_favoriteNumber is ignored.
617     aggregationcyclefreeness is ignored.
618     String is ignored.
619     Pet_nickName is ignored.
620     Person_nicknames is ignored.
621     Individual is ignored.
622     Person_fName is ignored.
623     Parenthood is ignored.
624     Integer is ignored.
625     Person_lName is ignored.
626     Individual_id is ignored.
627     Pet_id is ignored.
628     forbiddensharing is ignored.
629 Comparisons that classify as equal
630     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
631     String enabled value is both false.
632     Real enabled value is both false.
633     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
634     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
635     Invariant settings are empty.
636     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
637     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
638 Comparisons that classify as right prelimary overlapping
639     Integer_min and Integer_max classify as right is overlapping ( left:
             [ Integer_min=0 ; Integer_max=100 ] ; right: [ Integer_min=-10
            ; Integer_max=0 ] ).
640
641 21. config4 != config7 :
642 Comparisons that are ignored
643     Individual is ignored.
644     Person_nicknames is ignored.
645     Person_lName is ignored.
646     Person is ignored.
647     Pet is ignored.
648     Person_favoriteNumber is ignored.
649     aggregationcyclefreeness is ignored.
650     Person_id is ignored.
651     PetSitting is ignored.
652     Pet_id is ignored.
653     Pet_nickName is ignored.
654     Parenthood is ignored.
655     Person_yearB is ignored.
656     Person_fName is ignored.
657     Integer is ignored.
658     Individual_id is ignored.
659     forbiddensharing is ignored.
660     String is ignored.
661 Comparisons that classify as equal
```

```
662     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
663     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
664     Real enabled value is both false.
665     Invariant settings are empty.
666     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
667     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
668     String enabled value is both false.
669     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
670 Comparisons that classify as left prelimary overlapping
671     Integer_min and Integer_max classify as left is overlapping ( left:
            [ Integer_min=0 ; Integer_max=100 ] ; right: [ Integer_min=100 ;
             Integer_max=1000 ] ).
672
673 22. config4 != config8 :
674 Comparisons that are ignored
675     Person_yearB is ignored.
676     String is ignored.
677     Person_id is ignored.
678     Individual_id is ignored.
679     Integer is ignored.
680     forbiddensharing is ignored.
681     Person_favoriteNumber is ignored.
682     PetSitting is ignored.
683     Pet_id is ignored.
684     Pet_nickName is ignored.
685     aggregationcyclefreeness is ignored.
686     Pet is ignored.
687     Person_lName is ignored.
688     Individual is ignored.
689     Parenthood is ignored.
690     Person is ignored.
691     Person_nicknames is ignored.
692     Person_fName is ignored.
693 Comparisons that classify as equal
694     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
695     Real enabled value is both false.
696     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
697     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
698     String enabled value is both false.
699     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
700     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
701     Invariant settings are empty.
702 Comparisons that classify as right disjoint
703     Integer_min and Integer_max classify as right disjoint ( left: [
            Integer_min=0 ; Integer_max=100 ] ; right: [ Integer_min=-100000
             ; Integer_max=-10000 ] ).
704
```

```
23. config5 != config6 :
Comparisons that are ignored
   Person is ignored.
   Parenthood is ignored.
   Person_nicknames is ignored.
   String is ignored.
   Integer is ignored.
   Person_id is ignored.
   Person_lName is ignored.
   aggregationcyclefreeness is ignored.
   Pet_nickName is ignored.
   Individual_id is ignored.
   Pet_id is ignored.
   Person_yearB is ignored.
   Person_favoriteNumber is ignored.
   Individual is ignored.
   PetSitting is ignored.
   Person_fName is ignored.
   forbiddensharing is ignored.
   Pet is ignored.
Comparisons that classify as equal
   Parenthood_min and Parenthood_max classify equality ( left: [
       Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
       =1 ; Parenthood_max=1 ] ).
   PetSitting_min and PetSitting_max classify equality ( left: [
       PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
       =1 ; PetSitting_max=1 ] ).
   String enabled value is both false.
   Real enabled value is both false.
   Person_min and Person_max classify equality ( left: [ Person_min=1 ;
        Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
   Individual_min and Individual_max classify equality ( left: [
       Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
       =1 ; Individual_max=1 ] ).
   Invariant settings are empty.
   Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
       =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
Comparisons that classify as right prelimary overlapping
   Integer_min and Integer_max classify as right is overlapping ( left:
        [ Integer_min=0 ; Integer_max=1000 ] ; right: [ Integer_min=-10
        ; Integer_max=0 ] ).

24. config5 >= config7 :
Comparisons that are ignored
   Pet_id is ignored.
   Person_fName is ignored.
   PetSitting is ignored.
   Parenthood is ignored.
   Individual_id is ignored.
   Person_lName is ignored.
   String is ignored.
   Pet is ignored.
   forbiddensharing is ignored.
   Person_yearB is ignored.
   Individual is ignored.
   Person_id is ignored.
   aggregationcyclefreeness is ignored.
   Person_nicknames is ignored.
   Pet_nickName is ignored.
   Person is ignored.
   Integer is ignored.
   Person_favoriteNumber is ignored.
Comparisons that classify as equal
```

```
758     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
            Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
759     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
760     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
761     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
762     String enabled value is both false.
763     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
764     Real enabled value is both false.
765     Invariant settings are empty.
766   Comparisons that classify as left is broad
767     Integer_min and Integer_max classify as left is broad ( left: [
            Integer_min=0 ; Integer_max=1000 ] ; right: [ Integer_min=100 ;
            Integer_max=1000 ] ).
768
769   25. config5 != config8 :
770   Comparisons that are ignored
771     Person is ignored.
772     Person_yearB is ignored.
773     Person_lName is ignored.
774     Pet_id is ignored.
775     Person_nicknames is ignored.
776     aggregationcyclefreeness is ignored.
777     Integer is ignored.
778     Pet_nickName is ignored.
779     forbiddensharing is ignored.
780     Parenthood is ignored.
781     Pet is ignored.
782     Individual_id is ignored.
783     Person_favoriteNumber is ignored.
784     PetSitting is ignored.
785     Individual is ignored.
786     String is ignored.
787     Person_id is ignored.
788     Person_fName is ignored.
789   Comparisons that classify as equal
790     Real enabled value is both false.
791     Invariant settings are empty.
792     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
793     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
794     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
795     String enabled value is both false.
796     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
            Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
797     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
798   Comparisons that classify as right disjoint
799     Integer_min and Integer_max classify as right disjoint ( left: [
            Integer_min=0 ; Integer_max=1000 ] ; right: [ Integer_min
            =-100000 ; Integer_max=-10000 ] ).
800
```

```
801 26. config6 != config7 :
802 Comparisons that are ignored
803     String is ignored.
804     PetSitting is ignored.
805     Individual_id is ignored.
806     Person_lName is ignored.
807     aggregationcyclefreeness is ignored.
808     Person_nicknames is ignored.
809     forbiddensharing is ignored.
810     Parenthood is ignored.
811     Person_fName is ignored.
812     Person_yearB is ignored.
813     Person_id is ignored.
814     Pet_id is ignored.
815     Person is ignored.
816     Individual is ignored.
817     Integer is ignored.
818     Pet is ignored.
819     Person_favoriteNumber is ignored.
820     Pet_nickName is ignored.
821 Comparisons that classify as equal
822     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
823     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
824     String enabled value is both false.
825     Invariant settings are empty.
826     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
827     Real enabled value is both false.
828     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
829     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
830 Comparisons that classify as left disjoint
831     Integer_min and Integer_max classify as left disjoint ( left: [
            Integer_min=-10 ; Integer_max=0 ] ; right: [ Integer_min=100 ;
            Integer_max=1000 ] ).
832
833 27. config6 != config8 :
834 Comparisons that are ignored
835     Person_nicknames is ignored.
836     Individual is ignored.
837     Pet_nickName is ignored.
838     Pet_id is ignored.
839     Person_lName is ignored.
840     forbiddensharing is ignored.
841     Person is ignored.
842     Integer is ignored.
843     Person_yearB is ignored.
844     aggregationcyclefreeness is ignored.
845     String is ignored.
846     Person_id is ignored.
847     PetSitting is ignored.
848     Individual_id is ignored.
849     Parenthood is ignored.
850     Person_fName is ignored.
851     Person_favoriteNumber is ignored.
852     Pet is ignored.
853 Comparisons that classify as equal
```

```
854     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
855     String enabled value is both false.
856     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
857     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
858     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
859     Invariant settings are empty.
860     Real enabled value is both false.
861     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
862   Comparisons that classify as right disjoint
863     Integer_min and Integer_max classify as right disjoint ( left: [
            Integer_min=-10 ; Integer_max=0 ] ; right: [ Integer_min=-100000
             ; Integer_max=-10000 ] ).
864
865   28. config7 != config8 :
866   Comparisons that are ignored
867     aggregationcyclefreeness is ignored.
868     Individual_id is ignored.
869     Person_lName is ignored.
870     Parenthood is ignored.
871     Integer is ignored.
872     Person is ignored.
873     Pet_id is ignored.
874     Person_id is ignored.
875     Pet_nickName is ignored.
876     Person_favoriteNumber is ignored.
877     Individual is ignored.
878     PetSitting is ignored.
879     forbiddensharing is ignored.
880     Pet is ignored.
881     String is ignored.
882     Person_nicknames is ignored.
883     Person_fName is ignored.
884     Person_yearB is ignored.
885   Comparisons that classify as equal
886     PetSitting_min and PetSitting_max classify equality ( left: [
            PetSitting_min=1 ; PetSitting_max=1 ] ; right: [ PetSitting_min
            =1 ; PetSitting_max=1 ] ).
887     Individual_min and Individual_max classify equality ( left: [
            Individual_min=1 ; Individual_max=1 ] ; right: [ Individual_min
            =1 ; Individual_max=1 ] ).
888     Parenthood_min and Parenthood_max classify equality ( left: [
            Parenthood_min=1 ; Parenthood_max=1 ] ; right: [ Parenthood_min
            =1 ; Parenthood_max=1 ] ).
889     Person_min and Person_max classify equality ( left: [ Person_min=1 ;
             Person_max=1 ] ; right: [ Person_min=1 ; Person_max=1 ] ).
890     String enabled value is both false.
891     Pet_min and Pet_max classify equality ( left: [ Pet_min=1 ; Pet_max
            =1 ] ; right: [ Pet_min=1 ; Pet_max=1 ] ).
892     Invariant settings are empty.
893     Real enabled value is both false.
894   Comparisons that classify as right disjoint
895     Integer_min and Integer_max classify as right disjoint ( left: [
            Integer_min=100 ; Integer_max=1000 ] ; right: [ Integer_min
            =-100000 ; Integer_max=-10000 ] ).
896
```

```
897
898 Overview:
899
900 |   1 | <= | config1 | config2 |
901 |   2 | >= | config1 | config3 |
902 |   3 | != | config1 | config4 |
903 |   4 | != | config1 | config5 |
904 |   5 | >= | config1 | config6 |
905 |   6 | != | config1 | config7 |
906 |   7 | != | config1 | config8 |
907 |   8 | >= | config2 | config3 |
908 |   9 | >= | config2 | config4 |
909 |  10 | != | config2 | config5 |
910 |  11 | >= | config2 | config6 |
911 |  12 | != | config2 | config7 |
912 |  13 | != | config2 | config8 |
913 |  14 | <= | config3 | config4 |
914 |  15 | <= | config3 | config5 |
915 |  16 | != | config3 | config6 |
916 |  17 | != | config3 | config7 |
917 |  18 | != | config3 | config8 |
918 |  19 | <= | config4 | config5 |
919 |  20 | != | config4 | config6 |
920 |  21 | != | config4 | config7 |
921 |  22 | != | config4 | config8 |
922 |  23 | != | config5 | config6 |
923 |  24 | >= | config5 | config7 |
924 |  25 | != | config5 | config8 |
925 |  26 | != | config6 | config7 |
926 |  27 | != | config6 | config8 |
927 |  28 | != | config7 | config8 |
928
929 The comparison shows 1 family with more than two members:
930 1: [config2, config3, config4, config5, config6, config7, config1]
```

LISTING A.1: UML-based Specification Environment command-line interface output for comparing configurations from listing 3.1

Appendix B

Formal proof: Arbitrary operands order for comparison results merging

In the following it is proofed that the order of to be merged types is arbitrary when merging configuration comparison partial comparison result types.

Proof via mathematical induction:
It must be proofed, that the result of the n-ary merge is independent of the order of the selected operands for the application of the underlying binary merge. Let $nmerge(o_1, \ldots, o_n)$ be a function with $n \geq 2$ arguments where $o_1, \ldots, o_n \in O$ are symbols from table 3.1. Let $nmerge(o_1, \ldots, o_n) = nmerge(nmerge(o_1, \ldots, o_{n-1}), o_n)$ be the recursive definition of the function for $n \geq 3$. Let the result of $nmerge(o_1, o_2)$ be defined in table 3.2.

Base step:
It must be proved that the statement is valid for $n = 2$, the smallest number. It must hold $\forall o_1, o_2 \in O : nmerge(o_1, o_2) = nmerge(o_2, o_1)$. This holds since the result of $nmerge(o_1, o_2)$ and $nmerge(o_2, o_1)$ are defined in the inverted matrix in table 3.2. For $n = 2$ the function is therefor independent of the sequence of the operands.

Induction step:
It will be proved that if the statement applies to a certain number n, the statement also applies to $n + 1$, the next larger number. It must hold $\forall o_1, ..., o_{n-1}, o_n, o_{n+1} \in O$:

$$\begin{aligned} nmerge(nmerge(o_1, ..., o_n), o_{n+1}) &= nmerge(nmerge(nmerge(o_1, ..., o_{n-1}), o_n), o_{n+1}) \\ &= nmerge(nmerge(o_n, nmerge(o_1, ..., o_{n-1})), o_{n+1}) \\ &= nmerge(nmerge(o_{n+1}, o_n), nmerge(o_1, ..., o_{n-1})) \\ &= nmerge(nmerge(o_n, o_{n+1}), nmerge(o_1, ..., o_{n-1})) \\ &= nmerge(nmerge(nmerge(o_1, ..., o_{n-1}), o_{n+1}), o_n) \\ &= nmerge(nmerge(o_{n+1}, nmerge(o_1, ..., o_{n-1})), o_n) \end{aligned} \tag{B.1}$$

This is formulated so that for $n + 1$ the property can only be fulfilled if it is fulfilled for n. Regarding $n = 3$, all parts $nmerge(o_1, ..., o_{n-1})$ are independent of the sequence of the operands, which is shown in the base step. So when B.1 holds for $n = 3$ when it holds for $n = 2$, it also holds for $n + 1$ when it holds for n because when it holds for $n = 3$, then it also holds for $n = 4$ and so on. Therefor, it is to be proofed that B.1 holds for $n = 3$. It is to be shown that for all possible combinations it holds that the

result is independent of the order. It must hold $\forall o_1, o_2, o_3 \in O$:

$$\begin{aligned} nmerge(o_1, o_2, o_3) &= nmerge(nmerge(o_1, o_2), o_3) \\ &= nmerge(nmerge(o_1, o_3), o_2) \\ &= nmerge(nmerge(o_2, o_1), o_3) \\ &= nmerge(nmerge(o_2, o_3), o_1) \\ &= nmerge(nmerge(o_3, o_1), o_2) \\ &= nmerge(nmerge(o_3, o_2), o_1) \end{aligned} \tag{B.2}$$

Since $nmerge(nmerge(o_1, o_3), o_2) = nmerge(nmerge(o_3, o_1), o_2)$, $nmerge(nmerge(o_1, o_2), o_3) = nmerge(nmerge(o_2, o_1), o_3)$ and $nmerge(nmerge(o_2, o_3), o_1) = nmerge(nmerge(o_3, o_2), o_1)$ already hold because of the order independence, only $\forall o_1, o_2, o_3 \in O$:

$$\begin{aligned} nmerge(o_1, o_2, o_3) &= nmerge(nmerge(o_1, o_2), o_3) \\ &= nmerge(nmerge(o_1, o_3), o_2) \\ &= nmerge(nmerge(o_2, o_3), o_1) \end{aligned} \tag{B.3}$$

must hold. This is shown in the following for all variants of $o_1, o_2, o_3 \in O$.

When all three operands are equal the result is the same as the operands.

When two of the three operands are equal it is not obvious that here too the result is independent of the operands order. However, this can be shown by case distinction for all $o_1, o_2 \in O$, where $o_1 \neq o_2$. It must hold $\forall \{o_1, o_2\} \subset O$:

$$\begin{aligned} nmerge(nmerge(o_1, o_1), o_2) &= nmerge(nmerge(o_2, o_1), o_1) \\ &= nmerge(nmerge(o_1, o_2), o_1) \\ &= nmerge(nmerge(o_2, o_2), o_1) \\ &= nmerge(nmerge(o_1, o_2), o_2) \\ &= nmerge(nmerge(o_2, o_1), o_2) \end{aligned} \tag{B.4}$$

Since $nmerge(nmerge(o_2, o_1), o_1) = nmerge(nmerge(o_1, o_2), o_1)$ and $nmerge(nmerge(o_1, o_2), o_2) = nmerge(nmerge(o_2, o_1), o_2)$ already holds because of the order independence, only the following must hold $\forall \{o_1, o_2\} \subset O$:

$$\begin{aligned} nmerge(nmerge(o_1, o_1), o_2) &= nmerge(nmerge(o_2, o_1), o_1) \\ &= nmerge(nmerge(o_2, o_2), o_1) \\ &= nmerge(nmerge(o_1, o_2), o_2) \end{aligned} \tag{B.5}$$

$\{$`!?`, `!!`$\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!?}), \texttt{!!}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{!?}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{!!}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{!!}), \texttt{!!}) \end{aligned} \tag{B.6}$$

$\{$`!?`, `..`$\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!?}), \texttt{..}) \\ &= nmerge(nmerge(\texttt{..}, \texttt{!?}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{..}, \texttt{..}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{..}), \texttt{..}) \end{aligned} \tag{B.7}$$

$\{\texttt{!?}, \texttt{!=}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!?}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{!?}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{!=}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{!=}), \texttt{!=}) \end{aligned} \tag{B.8}$$

$\{\texttt{!?}, \texttt{==}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!?}), \texttt{==}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{!?}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{==}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{==}), \texttt{==}) \end{aligned} \tag{B.9}$$

$\{\texttt{!?}, \texttt{>=}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!?}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{>=}, \texttt{!?}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{>=}, \texttt{>=}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{>=}), \texttt{>=}) \end{aligned} \tag{B.10}$$

$\{\texttt{!?}, \texttt{<=}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!?}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{<=}, \texttt{!?}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{<=}, \texttt{<=}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{<=}), \texttt{<=}) \end{aligned} \tag{B.11}$$

$\{\texttt{!?}, \texttt{lO}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!?}), \texttt{lO}) \\ &= nmerge(nmerge(\texttt{lO}, \texttt{!?}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{lO}, \texttt{lO}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{lO}), \texttt{lO}) \end{aligned} \tag{B.12}$$

$\{\texttt{!?}, \texttt{rO}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!?}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{rO}, \texttt{!?}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{rO}, \texttt{rO}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{rO}), \texttt{rO}) \end{aligned} \tag{B.13}$$

$\{\texttt{!?}, \texttt{lD}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!?}), \texttt{lD}) \\ &= nmerge(nmerge(\texttt{lD}, \texttt{!?}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{lD}, \texttt{lD}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{lD}), \texttt{lD}) \end{aligned} \tag{B.14}$$

$\{\texttt{!?}, \texttt{rD}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!?}), \texttt{rD}) \\ &= nmerge(nmerge(\texttt{rD}, \texttt{!?}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{rD}, \texttt{rD}), \texttt{!?}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{rD}), \texttt{rD}) \end{aligned} \tag{B.15}$$

$\{!!, ..\}$

$$\begin{aligned} !! &= nmerge(nmerge(!!, !!), ..) \\ &= nmerge(nmerge(.., !!), !!) \\ &= nmerge(nmerge(.., ..), !!) \\ &= nmerge(nmerge(!!, ..), ..) \end{aligned} \tag{B.16}$$

$\{!!, !=\}$

$$\begin{aligned} !! &= nmerge(nmerge(!!, !!), !=) \\ &= nmerge(nmerge(!=, !!), !!) \\ &= nmerge(nmerge(!=, !=), !!) \\ &= nmerge(nmerge(!!, !=), !=) \end{aligned} \tag{B.17}$$

$\{!!, ==\}$

$$\begin{aligned} !! &= nmerge(nmerge(!!, !!), ==) \\ &= nmerge(nmerge(==, !!), !!) \\ &= nmerge(nmerge(==, ==), !!) \\ &= nmerge(nmerge(!!, ==), ==) \end{aligned} \tag{B.18}$$

$\{!!, >=\}$

$$\begin{aligned} !! &= nmerge(nmerge(!!, !!), >=) \\ &= nmerge(nmerge(>=, !!), !!) \\ &= nmerge(nmerge(>=, >=), !!) \\ &= nmerge(nmerge(!!, >=), >=) \end{aligned} \tag{B.19}$$

$\{!!, <=\}$

$$\begin{aligned} !! &= nmerge(nmerge(!!, !!), <=) \\ &= nmerge(nmerge(<=, !!), !!) \\ &= nmerge(nmerge(<=, <=), !!) \\ &= nmerge(nmerge(!!, <=), <=) \end{aligned} \tag{B.20}$$

$\{!!, \mathtt{lO}\}$

$$\begin{aligned} !! &= nmerge(nmerge(!!, !!), \mathtt{lO}) \\ &= nmerge(nmerge(\mathtt{lO}, !!), !!) \\ &= nmerge(nmerge(\mathtt{lO}, \mathtt{lO}), !!) \\ &= nmerge(nmerge(!!, \mathtt{lO}), \mathtt{lO}) \end{aligned} \tag{B.21}$$

$\{!!, \mathtt{rO}\}$

$$\begin{aligned} !! &= nmerge(nmerge(!!, !!), \mathtt{rO}) \\ &= nmerge(nmerge(\mathtt{rO}, !!), !!) \\ &= nmerge(nmerge(\mathtt{rO}, \mathtt{rO}), !!) \\ &= nmerge(nmerge(!!, \mathtt{rO}), \mathtt{rO}) \end{aligned} \tag{B.22}$$

$\{!!, \mathtt{lD}\}$

$$\begin{aligned} !! &= nmerge(nmerge(!!, !!), \mathtt{lD}) \\ &= nmerge(nmerge(\mathtt{lD}, !!), !!) \\ &= nmerge(nmerge(\mathtt{lD}, \mathtt{lD}), !!) \\ &= nmerge(nmerge(!!, \mathtt{lD}), \mathtt{lD}) \end{aligned} \tag{B.23}$$

$\{$`!!`,`rD`$\}$

$$\begin{aligned}
\texttt{!!} &= nmerge(nmerge(\texttt{!!},\texttt{!!}),\texttt{rD}) \\
&= nmerge(nmerge(\texttt{rD},\texttt{!!}),\texttt{!!}) \\
&= nmerge(nmerge(\texttt{rD},\texttt{rD}),\texttt{!!}) \\
&= nmerge(nmerge(\texttt{!!},\texttt{rD}),\texttt{rD})
\end{aligned} \tag{B.24}$$

$\{$`..`,`!=`$\}$

$$\begin{aligned}
\texttt{!=} &= nmerge(nmerge(..,..),\texttt{!=}) \\
&= nmerge(nmerge(\texttt{!=},..),..) \\
&= nmerge(nmerge(\texttt{!=},\texttt{!=}),..) \\
&= nmerge(nmerge(..,\texttt{!=}),\texttt{!=})
\end{aligned} \tag{B.25}$$

$\{$`..`,`==`$\}$

$$\begin{aligned}
\texttt{==} &= nmerge(nmerge(..,..),\texttt{==}) \\
&= nmerge(nmerge(\texttt{==},..),..) \\
&= nmerge(nmerge(\texttt{==},\texttt{==}),..) \\
&= nmerge(nmerge(..,\texttt{==}),\texttt{==})
\end{aligned} \tag{B.26}$$

$\{$`..`,`>=`$\}$

$$\begin{aligned}
\texttt{>=} &= nmerge(nmerge(..,..),\texttt{>=}) \\
&= nmerge(nmerge(\texttt{>=},..),..) \\
&= nmerge(nmerge(\texttt{>=},\texttt{>=}),..) \\
&= nmerge(nmerge(..,\texttt{>=}),\texttt{>=})
\end{aligned} \tag{B.27}$$

$\{$`..`,`<=`$\}$

$$\begin{aligned}
\texttt{<=} &= nmerge(nmerge(..,..),\texttt{<=}) \\
&= nmerge(nmerge(\texttt{<=},..),..) \\
&= nmerge(nmerge(\texttt{<=},\texttt{<=}),..) \\
&= nmerge(nmerge(..,\texttt{<=}),\texttt{<=})
\end{aligned} \tag{B.28}$$

$\{$`..`,`lO`$\}$

$$\begin{aligned}
\texttt{lO} &= nmerge(nmerge(..,..),\texttt{lO}) \\
&= nmerge(nmerge(\texttt{lO},..),..) \\
&= nmerge(nmerge(\texttt{lO},\texttt{lO}),..) \\
&= nmerge(nmerge(..,\texttt{lO}),\texttt{lO})
\end{aligned} \tag{B.29}$$

$\{$`..`,`rO`$\}$

$$\begin{aligned}
\texttt{rO} &= nmerge(nmerge(..,..),\texttt{rO}) \\
&= nmerge(nmerge(\texttt{rO},..),..) \\
&= nmerge(nmerge(\texttt{rO},\texttt{rO}),..) \\
&= nmerge(nmerge(..,\texttt{rO}),\texttt{rO})
\end{aligned} \tag{B.30}$$

$\{$`..`,`lD`$\}$

$$\begin{aligned}
\texttt{lD} &= nmerge(nmerge(..,..),\texttt{lD}) \\
&= nmerge(nmerge(\texttt{lD},..),..) \\
&= nmerge(nmerge(\texttt{lD},\texttt{lD}),..) \\
&= nmerge(nmerge(..,\texttt{lD}),\texttt{lD})
\end{aligned} \tag{B.31}$$

$\{\texttt{..},\texttt{rD}\}$

$$\begin{aligned}\texttt{rD} &= nmerge(nmerge(\texttt{..},\texttt{..}),\texttt{rD})\\ &= nmerge(nmerge(\texttt{rD},\texttt{..}),\texttt{..})\\ &= nmerge(nmerge(\texttt{rD},\texttt{rD}),\texttt{..})\\ &= nmerge(nmerge(\texttt{..},\texttt{rD}),\texttt{rD})\end{aligned} \tag{B.32}$$

$\{\texttt{!=},\texttt{==}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{!=}),\texttt{==})\\ &= nmerge(nmerge(\texttt{==},\texttt{!=}),\texttt{!=})\\ &= nmerge(nmerge(\texttt{==},\texttt{==}),\texttt{!=})\\ &= nmerge(nmerge(\texttt{!=},\texttt{==}),\texttt{==})\end{aligned} \tag{B.33}$$

$\{\texttt{!=},\texttt{>=}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{!=}),\texttt{>=})\\ &= nmerge(nmerge(\texttt{>=},\texttt{!=}),\texttt{!=})\\ &= nmerge(nmerge(\texttt{>=},\texttt{>=}),\texttt{!=})\\ &= nmerge(nmerge(\texttt{!=},\texttt{>=}),\texttt{>=})\end{aligned} \tag{B.34}$$

$\{\texttt{!=},\texttt{<=}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{!=}),\texttt{<=})\\ &= nmerge(nmerge(\texttt{<=},\texttt{!=}),\texttt{!=})\\ &= nmerge(nmerge(\texttt{<=},\texttt{<=}),\texttt{!=})\\ &= nmerge(nmerge(\texttt{!=},\texttt{<=}),\texttt{<=})\end{aligned} \tag{B.35}$$

$\{\texttt{!=},\texttt{lO}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{!=}),\texttt{lO})\\ &= nmerge(nmerge(\texttt{lO},\texttt{!=}),\texttt{!=})\\ &= nmerge(nmerge(\texttt{lO},\texttt{lO}),\texttt{!=})\\ &= nmerge(nmerge(\texttt{!=},\texttt{lO}),\texttt{lO})\end{aligned} \tag{B.36}$$

$\{\texttt{!=},\texttt{rO}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{!=}),\texttt{rO})\\ &= nmerge(nmerge(\texttt{rO},\texttt{!=}),\texttt{!=})\\ &= nmerge(nmerge(\texttt{rO},\texttt{rO}),\texttt{!=})\\ &= nmerge(nmerge(\texttt{!=},\texttt{rO}),\texttt{rO})\end{aligned} \tag{B.37}$$

$\{\texttt{!=},\texttt{lD}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{!=}),\texttt{lD})\\ &= nmerge(nmerge(\texttt{lD},\texttt{!=}),\texttt{!=})\\ &= nmerge(nmerge(\texttt{lD},\texttt{lD}),\texttt{!=})\\ &= nmerge(nmerge(\texttt{!=},\texttt{lD}),\texttt{lD})\end{aligned} \tag{B.38}$$

$\{\texttt{!=},\texttt{rD}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{!=}),\texttt{rD})\\ &= nmerge(nmerge(\texttt{rD},\texttt{!=}),\texttt{!=})\\ &= nmerge(nmerge(\texttt{rD},\texttt{rD}),\texttt{!=})\\ &= nmerge(nmerge(\texttt{!=},\texttt{rD}),\texttt{rD})\end{aligned} \tag{B.39}$$

$\{\texttt{==},\texttt{>=}\}$

$$\begin{aligned}\texttt{==} &= nmerge(nmerge(\texttt{==},\texttt{==}),\texttt{>=})\\ &= nmerge(nmerge(\texttt{>=},\texttt{==}),\texttt{==})\\ &= nmerge(nmerge(\texttt{>=},\texttt{>=}),\texttt{==})\\ &= nmerge(nmerge(\texttt{==},\texttt{>=}),\texttt{>=})\end{aligned} \tag{B.40}$$

$\{\texttt{==},\texttt{<=}\}$

$$\begin{aligned}\texttt{==} &= nmerge(nmerge(\texttt{==},\texttt{==}),\texttt{<=})\\ &= nmerge(nmerge(\texttt{<=},\texttt{==}),\texttt{==})\\ &= nmerge(nmerge(\texttt{<=},\texttt{<=}),\texttt{==})\\ &= nmerge(nmerge(\texttt{==},\texttt{<=}),\texttt{<=})\end{aligned} \tag{B.41}$$

$\{\texttt{==},\texttt{lO}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{==},\texttt{==}),\texttt{lO})\\ &= nmerge(nmerge(\texttt{lO},\texttt{==}),\texttt{==})\\ &= nmerge(nmerge(\texttt{lO},\texttt{lO}),\texttt{==})\\ &= nmerge(nmerge(\texttt{==},\texttt{lO}),\texttt{lO})\end{aligned} \tag{B.42}$$

$\{\texttt{==},\texttt{rO}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{==},\texttt{==}),\texttt{rO})\\ &= nmerge(nmerge(\texttt{rO},\texttt{==}),\texttt{==})\\ &= nmerge(nmerge(\texttt{rO},\texttt{rO}),\texttt{==})\\ &= nmerge(nmerge(\texttt{==},\texttt{rO}),\texttt{rO})\end{aligned} \tag{B.43}$$

$\{\texttt{==},\texttt{lD}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{==},\texttt{==}),\texttt{lD})\\ &= nmerge(nmerge(\texttt{lD},\texttt{==}),\texttt{==})\\ &= nmerge(nmerge(\texttt{lD},\texttt{lD}),\texttt{==})\\ &= nmerge(nmerge(\texttt{==},\texttt{lD}),\texttt{lD})\end{aligned} \tag{B.44}$$

$\{\texttt{==},\texttt{rD}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{==},\texttt{==}),\texttt{rD})\\ &= nmerge(nmerge(\texttt{rD},\texttt{==}),\texttt{==})\\ &= nmerge(nmerge(\texttt{rD},\texttt{rD}),\texttt{==})\\ &= nmerge(nmerge(\texttt{==},\texttt{rD}),\texttt{rD})\end{aligned} \tag{B.45}$$

$\{\texttt{>=},\texttt{<=}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{>=},\texttt{>=}),\texttt{<=})\\ &= nmerge(nmerge(\texttt{<=},\texttt{>=}),\texttt{>=})\\ &= nmerge(nmerge(\texttt{<=},\texttt{<=}),\texttt{>=})\\ &= nmerge(nmerge(\texttt{>=},\texttt{<=}),\texttt{<=})\end{aligned} \tag{B.46}$$

$\{\texttt{>=},\texttt{lO}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{>=},\texttt{>=}),\texttt{lO})\\ &= nmerge(nmerge(\texttt{lO},\texttt{>=}),\texttt{>=})\\ &= nmerge(nmerge(\texttt{lO},\texttt{lO}),\texttt{>=})\\ &= nmerge(nmerge(\texttt{>=},\texttt{lO}),\texttt{lO})\end{aligned} \tag{B.47}$$

$\{\texttt{>=}, \texttt{rO}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{>=}, \texttt{>=}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{rO}, \texttt{>=}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{rO}, \texttt{rO}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{>=}, \texttt{rO}), \texttt{rO}) \end{aligned} \tag{B.48}$$

$\{\texttt{>=}, \texttt{lD}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{>=}, \texttt{>=}), \texttt{lD}) \\ &= nmerge(nmerge(\texttt{lD}, \texttt{>=}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{lD}, \texttt{lD}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{>=}, \texttt{lD}), \texttt{lD}) \end{aligned} \tag{B.49}$$

$\{\texttt{>=}, \texttt{rD}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{>=}, \texttt{>=}), \texttt{rD}) \\ &= nmerge(nmerge(\texttt{rD}, \texttt{>=}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{rD}, \texttt{rD}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{>=}, \texttt{rD}), \texttt{rD}) \end{aligned} \tag{B.50}$$

$\{\texttt{<=}, \texttt{lO}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{<=}, \texttt{<=}), \texttt{lO}) \\ &= nmerge(nmerge(\texttt{lO}, \texttt{<=}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{lO}, \texttt{lO}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{<=}, \texttt{lO}), \texttt{lO}) \end{aligned} \tag{B.51}$$

$\{\texttt{<=}, \texttt{rO}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{<=}, \texttt{<=}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{rO}, \texttt{<=}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{rO}, \texttt{rO}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{<=}, \texttt{rO}), \texttt{rO}) \end{aligned} \tag{B.52}$$

$\{\texttt{<=}, \texttt{lD}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{<=}, \texttt{<=}), \texttt{lD}) \\ &= nmerge(nmerge(\texttt{lD}, \texttt{<=}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{lD}, \texttt{lD}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{<=}, \texttt{lD}), \texttt{lD}) \end{aligned} \tag{B.53}$$

$\{\texttt{<=}, \texttt{rD}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{<=}, \texttt{<=}), \texttt{rD}) \\ &= nmerge(nmerge(\texttt{rD}, \texttt{<=}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{rD}, \texttt{rD}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{<=}, \texttt{rD}), \texttt{rD}) \end{aligned} \tag{B.54}$$

$\{\texttt{lO}, \texttt{rO}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{lO}, \texttt{lO}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{rO}, \texttt{lO}), \texttt{lO}) \\ &= nmerge(nmerge(\texttt{rO}, \texttt{rO}), \texttt{lO}) \\ &= nmerge(nmerge(\texttt{lO}, \texttt{rO}), \texttt{rO}) \end{aligned} \tag{B.55}$$

$\{\texttt{lO}, \texttt{lD}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{lO}, \texttt{lO}), \texttt{lD}) \\ &= nmerge(nmerge(\texttt{lD}, \texttt{lO}), \texttt{lO}) \\ &= nmerge(nmerge(\texttt{lD}, \texttt{lD}), \texttt{lO}) \\ &= nmerge(nmerge(\texttt{lO}, \texttt{lD}), \texttt{lD}) \end{aligned} \tag{B.56}$$

$\{\texttt{lO}, \texttt{rD}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{lO}, \texttt{lO}), \texttt{rD}) \\ &= nmerge(nmerge(\texttt{rD}, \texttt{lO}), \texttt{lO}) \\ &= nmerge(nmerge(\texttt{rD}, \texttt{rD}), \texttt{lO}) \\ &= nmerge(nmerge(\texttt{lO}, \texttt{rD}), \texttt{rD}) \end{aligned} \tag{B.57}$$

$\{\texttt{rO}, \texttt{lD}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{rO}, \texttt{rO}), \texttt{lD}) \\ &= nmerge(nmerge(\texttt{lD}, \texttt{rO}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{lD}, \texttt{lD}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{rO}, \texttt{lD}), \texttt{lD}) \end{aligned} \tag{B.58}$$

$\{\texttt{rO}, \texttt{rD}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{rO}, \texttt{rO}), \texttt{rD}) \\ &= nmerge(nmerge(\texttt{rD}, \texttt{rO}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{rD}, \texttt{rD}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{rO}, \texttt{rD}), \texttt{rD}) \end{aligned} \tag{B.59}$$

$\{\texttt{lD}, \texttt{rD}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{lD}, \texttt{lD}), \texttt{rD}) \\ &= nmerge(nmerge(\texttt{rD}, \texttt{lD}), \texttt{lD}) \\ &= nmerge(nmerge(\texttt{rD}, \texttt{rD}), \texttt{lD}) \\ &= nmerge(nmerge(\texttt{lD}, \texttt{rD}), \texttt{rD}) \end{aligned} \tag{B.60}$$

When all three operands are different (all remaining cases) it is also not obvious that the result is independent of the operands order. However, this can be shown by case distinction for all $o_1, o_2, o_3 \in O$, where $o_1 \neq o_2$, $o_1 \neq o_3$ and $o_2 \neq o_3$. It must hold $\forall\{o_1, o_2, o_3\} \subset O$:

$$\begin{aligned} nmerge(o_1, o_2, o_3) &= nmerge(nmerge(o_1, o_2), o_3) \\ &= nmerge(nmerge(o_1, o_3), o_2) \\ &= nmerge(nmerge(o_2, o_3), o_1) \end{aligned} \tag{B.61}$$

$\{\texttt{!?}, \texttt{!!}, \texttt{..}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!!}), \texttt{..}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{..}), \texttt{!!}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{..}), \texttt{!?}) \end{aligned} \tag{B.62}$$

$\{\texttt{!?}, \texttt{!!}, \texttt{!=}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!!}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{!=}), \texttt{!!}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{!=}), \texttt{!?}) \end{aligned} \tag{B.63}$$

$\{$`!?`, `!!`, `==`$\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!!}), \texttt{==}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{==}), \texttt{!!}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{==}), \texttt{!?}) \end{aligned} \tag{B.64}$$

$\{$`!?`, `!!`, `>=`$\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!!}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{>=}), \texttt{!!}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{>=}), \texttt{!?}) \end{aligned} \tag{B.65}$$

$\{$`!?`, `!!`, `<=`$\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!!}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{<=}), \texttt{!!}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{<=}), \texttt{!?}) \end{aligned} \tag{B.66}$$

$\{$`!?`, `!!`, `lO`$\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!!}), \texttt{lO}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{lO}), \texttt{!!}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{lO}), \texttt{!?}) \end{aligned} \tag{B.67}$$

$\{$`!?`, `!!`, `rO`$\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!!}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{rO}), \texttt{!!}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{rO}), \texttt{!?}) \end{aligned} \tag{B.68}$$

$\{$`!?`, `!!`, `lD`$\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!!}), \texttt{lD}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{lD}), \texttt{!!}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{lD}), \texttt{!?}) \end{aligned} \tag{B.69}$$

$\{$`!?`, `..`, `!=`$\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{..}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{!=}), \texttt{..}) \\ &= nmerge(nmerge(\texttt{..}, \texttt{!=}), \texttt{!?}) \end{aligned} \tag{B.70}$$

$\{$`!?`, `..`, `==`$\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{..}), \texttt{==}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{==}), \texttt{..}) \\ &= nmerge(nmerge(\texttt{..}, \texttt{==}), \texttt{!?}) \end{aligned} \tag{B.71}$$

$\{$`!?`, `..`, `>=`$\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{..}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{>=}), \texttt{..}) \\ &= nmerge(nmerge(\texttt{..}, \texttt{>=}), \texttt{!?}) \end{aligned} \tag{B.72}$$

$\{$`!?`, `..`, `<=`$\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{..}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{<=}), \texttt{..}) \\ &= nmerge(nmerge(\texttt{..}, \texttt{<=}), \texttt{!?}) \end{aligned} \tag{B.73}$$

$\{\texttt{!?}, \texttt{..}, \texttt{lO}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{..}), \texttt{lO}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{lO}), \texttt{..}) \\ &= nmerge(nmerge(\texttt{..}, \texttt{lO}), \texttt{!?}) \end{aligned} \tag{B.74}$$

$\{\texttt{!?}, \texttt{..}, \texttt{rO}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{..}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{rO}), \texttt{..}) \\ &= nmerge(nmerge(\texttt{..}, \texttt{rO}), \texttt{!?}) \end{aligned} \tag{B.75}$$

$\{\texttt{!?}, \texttt{..}, \texttt{lD}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{..}), \texttt{lD}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{lD}), \texttt{..}) \\ &= nmerge(nmerge(\texttt{..}, \texttt{lD}), \texttt{!?}) \end{aligned} \tag{B.76}$$

$\{\texttt{!?}, \texttt{!=}, \texttt{==}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!=}), \texttt{==}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{==}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{==}), \texttt{!?}) \end{aligned} \tag{B.77}$$

$\{\texttt{!?}, \texttt{!=}, \texttt{>=}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!=}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{>=}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{>=}), \texttt{!?}) \end{aligned} \tag{B.78}$$

$\{\texttt{!?}, \texttt{!=}, \texttt{<=}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!=}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{<=}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{<=}), \texttt{!?}) \end{aligned} \tag{B.79}$$

$\{\texttt{!?}, \texttt{!=}, \texttt{lO}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!=}), \texttt{lO}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{lO}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{lO}), \texttt{!?}) \end{aligned} \tag{B.80}$$

$\{\texttt{!?}, \texttt{!=}, \texttt{rO}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!=}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{rO}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{rO}), \texttt{!?}) \end{aligned} \tag{B.81}$$

$\{\texttt{!?}, \texttt{!=}, \texttt{lD}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{!=}), \texttt{lD}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{lD}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{lD}), \texttt{!?}) \end{aligned} \tag{B.82}$$

$\{\texttt{!?}, \texttt{==}, \texttt{>=}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{==}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{>=}), \texttt{==}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{>=}), \texttt{!?}) \end{aligned} \tag{B.83}$$

$\{\texttt{!?}, \texttt{==}, \texttt{<=}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{==}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{<=}), \texttt{==}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{<=}), \texttt{!?}) \end{aligned} \tag{B.84}$$

$\{\texttt{!?}, \texttt{==}, \texttt{lO}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{==}), \texttt{lO}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{lO}), \texttt{==}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{lO}), \texttt{!?}) \end{aligned} \tag{B.85}$$

$\{\texttt{!?}, \texttt{==}, \texttt{rO}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{==}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{rO}), \texttt{==}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{rO}), \texttt{!?}) \end{aligned} \tag{B.86}$$

$\{\texttt{!?}, \texttt{==}, \texttt{lD}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{==}), \texttt{lD}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{lD}), \texttt{==}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{lD}), \texttt{!?}) \end{aligned} \tag{B.87}$$

$\{\texttt{!?}, \texttt{>=}, \texttt{<=}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{>=}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{<=}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{>=}, \texttt{<=}), \texttt{!?}) \end{aligned} \tag{B.88}$$

$\{\texttt{!?}, \texttt{>=}, \texttt{lO}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{>=}), \texttt{lO}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{lO}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{>=}, \texttt{lO}), \texttt{!?}) \end{aligned} \tag{B.89}$$

$\{\texttt{!?}, \texttt{>=}, \texttt{rO}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{>=}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{rO}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{>=}, \texttt{rO}), \texttt{!?}) \end{aligned} \tag{B.90}$$

$\{\texttt{!?}, \texttt{>=}, \texttt{lD}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{>=}), \texttt{lD}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{lD}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{>=}, \texttt{lD}), \texttt{!?}) \end{aligned} \tag{B.91}$$

$\{\texttt{!?}, \texttt{<=}, \texttt{lO}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{<=}), \texttt{lO}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{lO}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{<=}, \texttt{lO}), \texttt{!?}) \end{aligned} \tag{B.92}$$

$\{\texttt{!?}, \texttt{<=}, \texttt{rO}\}$

$$\begin{aligned} \texttt{!?} &= nmerge(nmerge(\texttt{!?}, \texttt{<=}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{!?}, \texttt{rO}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{<=}, \texttt{rO}), \texttt{!?}) \end{aligned} \tag{B.93}$$

$\{\texttt{!?},\texttt{<=},\texttt{lD}\}$

$$\begin{aligned}\texttt{!?} &= nmerge(nmerge(\texttt{!?},\texttt{<=}),\texttt{lD})\\ &= nmerge(nmerge(\texttt{!?},\texttt{lD}),\texttt{<=})\\ &= nmerge(nmerge(\texttt{<=},\texttt{lD}),\texttt{!?})\end{aligned} \tag{B.94}$$

$\{\texttt{!?},\texttt{lO},\texttt{rO}\}$

$$\begin{aligned}\texttt{!?} &= nmerge(nmerge(\texttt{!?},\texttt{lO}),\texttt{rO})\\ &= nmerge(nmerge(\texttt{!?},\texttt{rO}),\texttt{lO})\\ &= nmerge(nmerge(\texttt{lO},\texttt{rO}),\texttt{!?})\end{aligned} \tag{B.95}$$

$\{\texttt{!?},\texttt{lO},\texttt{lD}\}$

$$\begin{aligned}\texttt{!?} &= nmerge(nmerge(\texttt{!?},\texttt{lO}),\texttt{lD})\\ &= nmerge(nmerge(\texttt{!?},\texttt{lD}),\texttt{lO})\\ &= nmerge(nmerge(\texttt{lO},\texttt{lD}),\texttt{!?})\end{aligned} \tag{B.96}$$

$\{\texttt{!?},\texttt{rO},\texttt{lD}\}$

$$\begin{aligned}\texttt{!?} &= nmerge(nmerge(\texttt{!?},\texttt{rO}),\texttt{lD})\\ &= nmerge(nmerge(\texttt{!?},\texttt{lD}),\texttt{rO})\\ &= nmerge(nmerge(\texttt{rO},\texttt{lD}),\texttt{!?})\end{aligned} \tag{B.97}$$

$\{\texttt{!!},\texttt{..},\texttt{!=}\}$

$$\begin{aligned}\texttt{!!} &= nmerge(nmerge(\texttt{!!},\texttt{..}),\texttt{!=})\\ &= nmerge(nmerge(\texttt{!!},\texttt{!=}),\texttt{..})\\ &= nmerge(nmerge(\texttt{..},\texttt{!=}),\texttt{!!})\end{aligned} \tag{B.98}$$

$\{\texttt{!!},\texttt{..},\texttt{==}\}$

$$\begin{aligned}\texttt{!!} &= nmerge(nmerge(\texttt{!!},\texttt{..}),\texttt{==})\\ &= nmerge(nmerge(\texttt{!!},\texttt{==}),\texttt{..})\\ &= nmerge(nmerge(\texttt{..},\texttt{==}),\texttt{!!})\end{aligned} \tag{B.99}$$

$\{\texttt{!!},\texttt{..},\texttt{>=}\}$

$$\begin{aligned}\texttt{!!} &= nmerge(nmerge(\texttt{!!},\texttt{..}),\texttt{>=})\\ &= nmerge(nmerge(\texttt{!!},\texttt{>=}),\texttt{..})\\ &= nmerge(nmerge(\texttt{..},\texttt{>=}),\texttt{!!})\end{aligned} \tag{B.100}$$

$\{\texttt{!!},\texttt{..},\texttt{<=}\}$

$$\begin{aligned}\texttt{!!} &= nmerge(nmerge(\texttt{!!},\texttt{..}),\texttt{<=})\\ &= nmerge(nmerge(\texttt{!!},\texttt{<=}),\texttt{..})\\ &= nmerge(nmerge(\texttt{..},\texttt{<=}),\texttt{!!})\end{aligned} \tag{B.101}$$

$\{\texttt{!!},\texttt{..},\texttt{lO}\}$

$$\begin{aligned}\texttt{!!} &= nmerge(nmerge(\texttt{!!},\texttt{..}),\texttt{lO})\\ &= nmerge(nmerge(\texttt{!!},\texttt{lO}),\texttt{..})\\ &= nmerge(nmerge(\texttt{..},\texttt{lO}),\texttt{!!})\end{aligned} \tag{B.102}$$

$\{\texttt{!!},\texttt{..},\texttt{rO}\}$

$$\begin{aligned}\texttt{!!} &= nmerge(nmerge(\texttt{!!},\texttt{..}),\texttt{rO})\\ &= nmerge(nmerge(\texttt{!!},\texttt{rO}),\texttt{..})\\ &= nmerge(nmerge(\texttt{..},\texttt{rO}),\texttt{!!})\end{aligned} \tag{B.103}$$

$\{$`!!`, `..`, `lD`$\}$

$$\begin{aligned} \texttt{!!} &= nmerge(nmerge(\texttt{!!}, \texttt{..}), \texttt{lD}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{lD}), \texttt{..}) \\ &= nmerge(nmerge(\texttt{..}, \texttt{lD}), \texttt{!!}) \end{aligned} \tag{B.104}$$

$\{$`!!`, `!=`, `==`$\}$

$$\begin{aligned} \texttt{!!} &= nmerge(nmerge(\texttt{!!}, \texttt{!=}), \texttt{==}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{==}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{==}), \texttt{!!}) \end{aligned} \tag{B.105}$$

$\{$`!!`, `!=`, `>=`$\}$

$$\begin{aligned} \texttt{!!} &= nmerge(nmerge(\texttt{!!}, \texttt{!=}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{>=}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{>=}), \texttt{!!}) \end{aligned} \tag{B.106}$$

$\{$`!!`, `!=`, `<=`$\}$

$$\begin{aligned} \texttt{!!} &= nmerge(nmerge(\texttt{!!}, \texttt{!=}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{<=}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{<=}), \texttt{!!}) \end{aligned} \tag{B.107}$$

$\{$`!!`, `!=`, `lO`$\}$

$$\begin{aligned} \texttt{!!} &= nmerge(nmerge(\texttt{!!}, \texttt{!=}), \texttt{lO}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{lO}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{lO}), \texttt{!!}) \end{aligned} \tag{B.108}$$

$\{$`!!`, `!=`, `rO`$\}$

$$\begin{aligned} \texttt{!!} &= nmerge(nmerge(\texttt{!!}, \texttt{!=}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{rO}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{rO}), \texttt{!!}) \end{aligned} \tag{B.109}$$

$\{$`!!`, `!=`, `lD`$\}$

$$\begin{aligned} \texttt{!!} &= nmerge(nmerge(\texttt{!!}, \texttt{!=}), \texttt{lD}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{lD}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{lD}), \texttt{!!}) \end{aligned} \tag{B.110}$$

$\{$`!!`, `==`, `>=`$\}$

$$\begin{aligned} \texttt{!!} &= nmerge(nmerge(\texttt{!!}, \texttt{==}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{>=}), \texttt{==}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{>=}), \texttt{!!}) \end{aligned} \tag{B.111}$$

$\{$`!!`, `==`, `<=`$\}$

$$\begin{aligned} \texttt{!!} &= nmerge(nmerge(\texttt{!!}, \texttt{==}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{<=}), \texttt{==}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{<=}), \texttt{!!}) \end{aligned} \tag{B.112}$$

$\{$`!!`, `==`, `lO`$\}$

$$\begin{aligned} \texttt{!!} &= nmerge(nmerge(\texttt{!!}, \texttt{==}), \texttt{lO}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{lO}), \texttt{==}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{lO}), \texttt{!!}) \end{aligned} \tag{B.113}$$

$\{\texttt{!!},\texttt{==},\texttt{r0}\}$

$$\begin{aligned}\texttt{!!} &= nmerge(nmerge(\texttt{!!},\texttt{==}),\texttt{r0})\\ &= nmerge(nmerge(\texttt{!!},\texttt{r0}),\texttt{==})\\ &= nmerge(nmerge(\texttt{==},\texttt{r0}),\texttt{!!})\end{aligned} \tag{B.114}$$

$\{\texttt{!!},\texttt{==},\texttt{l0}\}$

$$\begin{aligned}\texttt{!!} &= nmerge(nmerge(\texttt{!!},\texttt{==}),\texttt{l0})\\ &= nmerge(nmerge(\texttt{!!},\texttt{l0}),\texttt{==})\\ &= nmerge(nmerge(\texttt{==},\texttt{l0}),\texttt{!!})\end{aligned} \tag{B.115}$$

$\{\texttt{!!},\texttt{>=},\texttt{<=}\}$

$$\begin{aligned}\texttt{!!} &= nmerge(nmerge(\texttt{!!},\texttt{>=}),\texttt{<=})\\ &= nmerge(nmerge(\texttt{!!},\texttt{<=}),\texttt{>=})\\ &= nmerge(nmerge(\texttt{>=},\texttt{<=}),\texttt{!!})\end{aligned} \tag{B.116}$$

$\{\texttt{!!},\texttt{>=},\texttt{lO}\}$

$$\begin{aligned}\texttt{!!} &= nmerge(nmerge(\texttt{!!},\texttt{>=}),\texttt{lO})\\ &= nmerge(nmerge(\texttt{!!},\texttt{lO}),\texttt{>=})\\ &= nmerge(nmerge(\texttt{>=},\texttt{lO}),\texttt{!!})\end{aligned} \tag{B.117}$$

$\{\texttt{!!},\texttt{>=},\texttt{r0}\}$

$$\begin{aligned}\texttt{!!} &= nmerge(nmerge(\texttt{!!},\texttt{>=}),\texttt{r0})\\ &= nmerge(nmerge(\texttt{!!},\texttt{r0}),\texttt{>=})\\ &= nmerge(nmerge(\texttt{>=},\texttt{r0}),\texttt{!!})\end{aligned} \tag{B.118}$$

$\{\texttt{!!},\texttt{>=},\texttt{l0}\}$

$$\begin{aligned}\texttt{!!} &= nmerge(nmerge(\texttt{!!},\texttt{>=}),\texttt{l0})\\ &= nmerge(nmerge(\texttt{!!},\texttt{l0}),\texttt{>=})\\ &= nmerge(nmerge(\texttt{>=},\texttt{l0}),\texttt{!!})\end{aligned} \tag{B.119}$$

$\{\texttt{!!},\texttt{<=},\texttt{lO}\}$

$$\begin{aligned}\texttt{!!} &= nmerge(nmerge(\texttt{!!},\texttt{<=}),\texttt{lO})\\ &= nmerge(nmerge(\texttt{!!},\texttt{lO}),\texttt{<=})\\ &= nmerge(nmerge(\texttt{<=},\texttt{lO}),\texttt{!!})\end{aligned} \tag{B.120}$$

$\{\texttt{!!},\texttt{<=},\texttt{r0}\}$

$$\begin{aligned}\texttt{!!} &= nmerge(nmerge(\texttt{!!},\texttt{<=}),\texttt{r0})\\ &= nmerge(nmerge(\texttt{!!},\texttt{r0}),\texttt{<=})\\ &= nmerge(nmerge(\texttt{<=},\texttt{r0}),\texttt{!!})\end{aligned} \tag{B.121}$$

$\{\texttt{!!},\texttt{<=},\texttt{l0}\}$

$$\begin{aligned}\texttt{!!} &= nmerge(nmerge(\texttt{!!},\texttt{<=}),\texttt{l0})\\ &= nmerge(nmerge(\texttt{!!},\texttt{l0}),\texttt{<=})\\ &= nmerge(nmerge(\texttt{<=},\texttt{l0}),\texttt{!!})\end{aligned} \tag{B.122}$$

$\{\texttt{!!},\texttt{lO},\texttt{r0}\}$

$$\begin{aligned}\texttt{!!} &= nmerge(nmerge(\texttt{!!},\texttt{lO}),\texttt{r0})\\ &= nmerge(nmerge(\texttt{!!},\texttt{r0}),\texttt{lO})\\ &= nmerge(nmerge(\texttt{lO},\texttt{r0}),\texttt{!!})\end{aligned} \tag{B.123}$$

$\{$`!!`, `lO`, `lD`$\}$

$$\begin{aligned} \texttt{!!} &= nmerge(nmerge(\texttt{!!}, \texttt{lO}), \texttt{lD}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{lD}), \texttt{lO}) \\ &= nmerge(nmerge(\texttt{lO}, \texttt{lD}), \texttt{!!}) \end{aligned} \tag{B.124}$$

$\{$`!!`, `rO`, `lD`$\}$

$$\begin{aligned} \texttt{!!} &= nmerge(nmerge(\texttt{!!}, \texttt{rO}), \texttt{lD}) \\ &= nmerge(nmerge(\texttt{!!}, \texttt{lD}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{rO}, \texttt{lD}), \texttt{!!}) \end{aligned} \tag{B.125}$$

$\{$`..`, `!=`, `==`$\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{..}, \texttt{!=}), \texttt{==}) \\ &= nmerge(nmerge(\texttt{..}, \texttt{==}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{==}), \texttt{..}) \end{aligned} \tag{B.126}$$

$\{$`..`, `!=`, `>=`$\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{..}, \texttt{!=}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{..}, \texttt{>=}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{>=}), \texttt{..}) \end{aligned} \tag{B.127}$$

$\{$`..`, `!=`, `<=`$\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{..}, \texttt{!=}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{..}, \texttt{<=}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{<=}), \texttt{..}) \end{aligned} \tag{B.128}$$

$\{$`..`, `!=`, `lO`$\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{..}, \texttt{!=}), \texttt{lO}) \\ &= nmerge(nmerge(\texttt{..}, \texttt{lO}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{lO}), \texttt{..}) \end{aligned} \tag{B.129}$$

$\{$`..`, `!=`, `rO`$\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{..}, \texttt{!=}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{..}, \texttt{rO}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{rO}), \texttt{..}) \end{aligned} \tag{B.130}$$

$\{$`..`, `!=`, `lD`$\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{..}, \texttt{!=}), \texttt{lD}) \\ &= nmerge(nmerge(\texttt{..}, \texttt{lD}), \texttt{!=}) \\ &= nmerge(nmerge(\texttt{!=}, \texttt{lD}), \texttt{..}) \end{aligned} \tag{B.131}$$

$\{$`..`, `==`, `>=`$\}$

$$\begin{aligned} \texttt{>=} &= nmerge(nmerge(\texttt{..}, \texttt{==}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{..}, \texttt{>=}), \texttt{==}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{>=}), \texttt{..}) \end{aligned} \tag{B.132}$$

$\{$`..`, `==`, `<=`$\}$

$$\begin{aligned} \texttt{<=} &= nmerge(nmerge(\texttt{..}, \texttt{==}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{..}, \texttt{<=}), \texttt{==}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{<=}), \texttt{..}) \end{aligned} \tag{B.133}$$

$\{..,$`==`$,$`lO`$\}$

$$\begin{aligned}
\texttt{!=} &= nmerge(nmerge(..,\texttt{==}),\texttt{lO}) \\
&= nmerge(nmerge(..,\texttt{lO}),\texttt{==}) \\
&= nmerge(nmerge(\texttt{==},\texttt{lO}),..)
\end{aligned} \tag{B.134}$$

$\{..,$`==`$,$`rO`$\}$

$$\begin{aligned}
\texttt{!=} &= nmerge(nmerge(..,\texttt{==}),\texttt{rO}) \\
&= nmerge(nmerge(..,\texttt{rO}),\texttt{==}) \\
&= nmerge(nmerge(\texttt{==},\texttt{rO}),..)
\end{aligned} \tag{B.135}$$

$\{..,$`==`$,$`lD`$\}$

$$\begin{aligned}
\texttt{!=} &= nmerge(nmerge(..,\texttt{==}),\texttt{lD}) \\
&= nmerge(nmerge(..,\texttt{lD}),\texttt{==}) \\
&= nmerge(nmerge(\texttt{==},\texttt{lD}),..)
\end{aligned} \tag{B.136}$$

$\{..,$`>=`$,$`<=`$\}$

$$\begin{aligned}
\texttt{!=} &= nmerge(nmerge(..,\texttt{>=}),\texttt{<=}) \\
&= nmerge(nmerge(..,\texttt{<=}),\texttt{>=}) \\
&= nmerge(nmerge(\texttt{>=},\texttt{<=}),..)
\end{aligned} \tag{B.137}$$

$\{..,$`>=`$,$`lO`$\}$

$$\begin{aligned}
\texttt{!=} &= nmerge(nmerge(..,\texttt{>=}),\texttt{lO}) \\
&= nmerge(nmerge(..,\texttt{lO}),\texttt{>=}) \\
&= nmerge(nmerge(\texttt{>=},\texttt{lO}),..)
\end{aligned} \tag{B.138}$$

$\{..,$`>=`$,$`rO`$\}$

$$\begin{aligned}
\texttt{!=} &= nmerge(nmerge(..,\texttt{>=}),\texttt{rO}) \\
&= nmerge(nmerge(..,\texttt{rO}),\texttt{>=}) \\
&= nmerge(nmerge(\texttt{>=},\texttt{rO}),..)
\end{aligned} \tag{B.139}$$

$\{..,$`>=`$,$`lD`$\}$

$$\begin{aligned}
\texttt{!=} &= nmerge(nmerge(..,\texttt{>=}),\texttt{lD}) \\
&= nmerge(nmerge(..,\texttt{lD}),\texttt{>=}) \\
&= nmerge(nmerge(\texttt{>=},\texttt{lD}),..)
\end{aligned} \tag{B.140}$$

$\{..,$`<=`$,$`lO`$\}$

$$\begin{aligned}
\texttt{!=} &= nmerge(nmerge(..,\texttt{<=}),\texttt{lO}) \\
&= nmerge(nmerge(..,\texttt{lO}),\texttt{<=}) \\
&= nmerge(nmerge(\texttt{<=},\texttt{lO}),..)
\end{aligned} \tag{B.141}$$

$\{..,$`<=`$,$`rO`$\}$

$$\begin{aligned}
\texttt{!=} &= nmerge(nmerge(..,\texttt{<=}),\texttt{rO}) \\
&= nmerge(nmerge(..,\texttt{rO}),\texttt{<=}) \\
&= nmerge(nmerge(\texttt{<=},\texttt{rO}),..)
\end{aligned} \tag{B.142}$$

$\{..,$`<=`$,$`lD`$\}$

$$\begin{aligned}
\texttt{!=} &= nmerge(nmerge(..,\texttt{<=}),\texttt{lD}) \\
&= nmerge(nmerge(..,\texttt{lD}),\texttt{<=}) \\
&= nmerge(nmerge(\texttt{<=},\texttt{lD}),..)
\end{aligned} \tag{B.143}$$

$\{..,\texttt{lO},\texttt{rO}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(..,\texttt{lO}),\texttt{rO}) \\ &= nmerge(nmerge(..,\texttt{rO}),\texttt{lO}) \\ &= nmerge(nmerge(\texttt{lO},\texttt{rO}),..)\end{aligned} \tag{B.144}$$

$\{..,\texttt{lO},\texttt{lD}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(..,\texttt{lO}),\texttt{lD}) \\ &= nmerge(nmerge(..,\texttt{lD}),\texttt{lO}) \\ &= nmerge(nmerge(\texttt{lO},\texttt{lD}),..)\end{aligned} \tag{B.145}$$

$\{..,\texttt{rO},\texttt{lD}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(..,\texttt{rO}),\texttt{lD}) \\ &= nmerge(nmerge(..,\texttt{lD}),\texttt{rO}) \\ &= nmerge(nmerge(\texttt{rO},\texttt{lD}),..)\end{aligned} \tag{B.146}$$

$\{\texttt{!=},\texttt{==},\texttt{>=}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{==}),\texttt{>=}) \\ &= nmerge(nmerge(\texttt{!=},\texttt{>=}),\texttt{==}) \\ &= nmerge(nmerge(\texttt{==},\texttt{>=}),\texttt{!=})\end{aligned} \tag{B.147}$$

$\{\texttt{!=},\texttt{==},\texttt{<=}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{==}),\texttt{<=}) \\ &= nmerge(nmerge(\texttt{!=},\texttt{<=}),\texttt{==}) \\ &= nmerge(nmerge(\texttt{==},\texttt{<=}),\texttt{!=})\end{aligned} \tag{B.148}$$

$\{\texttt{!=},\texttt{==},\texttt{lO}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{==}),\texttt{lO}) \\ &= nmerge(nmerge(\texttt{!=},\texttt{lO}),\texttt{==}) \\ &= nmerge(nmerge(\texttt{==},\texttt{lO}),\texttt{!=})\end{aligned} \tag{B.149}$$

$\{\texttt{!=},\texttt{==},\texttt{rO}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{==}),\texttt{rO}) \\ &= nmerge(nmerge(\texttt{!=},\texttt{rO}),\texttt{==}) \\ &= nmerge(nmerge(\texttt{==},\texttt{rO}),\texttt{!=})\end{aligned} \tag{B.150}$$

$\{\texttt{!=},\texttt{==},\texttt{lD}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{==}),\texttt{lD}) \\ &= nmerge(nmerge(\texttt{!=},\texttt{lD}),\texttt{==}) \\ &= nmerge(nmerge(\texttt{==},\texttt{lD}),\texttt{!=})\end{aligned} \tag{B.151}$$

$\{\texttt{!=},\texttt{>=},\texttt{<=}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{>=}),\texttt{<=}) \\ &= nmerge(nmerge(\texttt{!=},\texttt{<=}),\texttt{>=}) \\ &= nmerge(nmerge(\texttt{>=},\texttt{<=}),\texttt{!=})\end{aligned} \tag{B.152}$$

$\{\texttt{!=},\texttt{>=},\texttt{lO}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{>=}),\texttt{lO}) \\ &= nmerge(nmerge(\texttt{!=},\texttt{lO}),\texttt{>=}) \\ &= nmerge(nmerge(\texttt{>=},\texttt{lO}),\texttt{!=})\end{aligned} \tag{B.153}$$

$\{\texttt{!=},\texttt{>=},\texttt{r0}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{>=}),\texttt{r0})\\ &= nmerge(nmerge(\texttt{!=},\texttt{r0}),\texttt{>=})\\ &= nmerge(nmerge(\texttt{>=},\texttt{r0}),\texttt{!=})\end{aligned} \tag{B.154}$$

$\{\texttt{!=},\texttt{>=},\texttt{l0}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{>=}),\texttt{l0})\\ &= nmerge(nmerge(\texttt{!=},\texttt{l0}),\texttt{>=})\\ &= nmerge(nmerge(\texttt{>=},\texttt{l0}),\texttt{!=})\end{aligned} \tag{B.155}$$

$\{\texttt{!=},\texttt{<=},\texttt{l0}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{<=}),\texttt{l0})\\ &= nmerge(nmerge(\texttt{!=},\texttt{l0}),\texttt{<=})\\ &= nmerge(nmerge(\texttt{<=},\texttt{l0}),\texttt{!=})\end{aligned} \tag{B.156}$$

$\{\texttt{!=},\texttt{<=},\texttt{r0}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{<=}),\texttt{r0})\\ &= nmerge(nmerge(\texttt{!=},\texttt{r0}),\texttt{<=})\\ &= nmerge(nmerge(\texttt{<=},\texttt{r0}),\texttt{!=})\end{aligned} \tag{B.157}$$

$\{\texttt{!=},\texttt{<=},\texttt{l1}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{<=}),\texttt{l1})\\ &= nmerge(nmerge(\texttt{!=},\texttt{l1}),\texttt{<=})\\ &= nmerge(nmerge(\texttt{<=},\texttt{l1}),\texttt{!=})\end{aligned} \tag{B.158}$$

$\{\texttt{!=},\texttt{l0},\texttt{r0}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{l0}),\texttt{r0})\\ &= nmerge(nmerge(\texttt{!=},\texttt{r0}),\texttt{l0})\\ &= nmerge(nmerge(\texttt{l0},\texttt{r0}),\texttt{!=})\end{aligned} \tag{B.159}$$

$\{\texttt{!=},\texttt{l0},\texttt{l1}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{l0}),\texttt{l1})\\ &= nmerge(nmerge(\texttt{!=},\texttt{l1}),\texttt{l0})\\ &= nmerge(nmerge(\texttt{l0},\texttt{l1}),\texttt{!=})\end{aligned} \tag{B.160}$$

$\{\texttt{!=},\texttt{r0},\texttt{l1}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{!=},\texttt{r0}),\texttt{l1})\\ &= nmerge(nmerge(\texttt{!=},\texttt{l1}),\texttt{r0})\\ &= nmerge(nmerge(\texttt{r0},\texttt{l1}),\texttt{!=})\end{aligned} \tag{B.161}$$

$\{\texttt{==},\texttt{>=},\texttt{<=}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{==},\texttt{>=}),\texttt{<=})\\ &= nmerge(nmerge(\texttt{==},\texttt{<=}),\texttt{>=})\\ &= nmerge(nmerge(\texttt{>=},\texttt{<=}),\texttt{==})\end{aligned} \tag{B.162}$$

$\{\texttt{==},\texttt{>=},\texttt{l0}\}$

$$\begin{aligned}\texttt{!=} &= nmerge(nmerge(\texttt{==},\texttt{>=}),\texttt{l0})\\ &= nmerge(nmerge(\texttt{==},\texttt{l0}),\texttt{>=})\\ &= nmerge(nmerge(\texttt{>=},\texttt{l0}),\texttt{==})\end{aligned} \tag{B.163}$$

$\{\texttt{==}, \texttt{>=}, \texttt{r0}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{==}, \texttt{>=}), \texttt{r0}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{r0}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{>=}, \texttt{r0}), \texttt{==}) \end{aligned} \tag{B.164}$$

$\{\texttt{==}, \texttt{>=}, \texttt{l0}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{==}, \texttt{>=}), \texttt{l0}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{l0}), \texttt{>=}) \\ &= nmerge(nmerge(\texttt{>=}, \texttt{l0}), \texttt{==}) \end{aligned} \tag{B.165}$$

$\{\texttt{==}, \texttt{<=}, \texttt{l0}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{==}, \texttt{<=}), \texttt{l0}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{l0}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{<=}, \texttt{l0}), \texttt{==}) \end{aligned} \tag{B.166}$$

$\{\texttt{==}, \texttt{<=}, \texttt{r0}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{==}, \texttt{<=}), \texttt{r0}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{r0}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{<=}, \texttt{r0}), \texttt{==}) \end{aligned} \tag{B.167}$$

$\{\texttt{==}, \texttt{<=}, \texttt{l0}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{==}, \texttt{<=}), \texttt{l0}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{l0}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{<=}, \texttt{l0}), \texttt{==}) \end{aligned} \tag{B.168}$$

$\{\texttt{==}, \texttt{l0}, \texttt{r0}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{==}, \texttt{l0}), \texttt{r0}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{r0}), \texttt{l0}) \\ &= nmerge(nmerge(\texttt{l0}, \texttt{r0}), \texttt{==}) \end{aligned} \tag{B.169}$$

$\{\texttt{==}, \texttt{l0}, \texttt{l0}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{==}, \texttt{l0}), \texttt{l0}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{l0}), \texttt{l0}) \\ &= nmerge(nmerge(\texttt{l0}, \texttt{l0}), \texttt{==}) \end{aligned} \tag{B.170}$$

$\{\texttt{==}, \texttt{r0}, \texttt{l0}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{==}, \texttt{r0}), \texttt{l0}) \\ &= nmerge(nmerge(\texttt{==}, \texttt{l0}), \texttt{r0}) \\ &= nmerge(nmerge(\texttt{r0}, \texttt{l0}), \texttt{==}) \end{aligned} \tag{B.171}$$

$\{\texttt{>=}, \texttt{<=}, \texttt{l0}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{>=}, \texttt{<=}), \texttt{l0}) \\ &= nmerge(nmerge(\texttt{>=}, \texttt{l0}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{<=}, \texttt{l0}), \texttt{>=}) \end{aligned} \tag{B.172}$$

$\{\texttt{>=}, \texttt{<=}, \texttt{r0}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{>=}, \texttt{<=}), \texttt{r0}) \\ &= nmerge(nmerge(\texttt{>=}, \texttt{r0}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{<=}, \texttt{r0}), \texttt{>=}) \end{aligned} \tag{B.173}$$

$\{\texttt{>=}, \texttt{<=}, \texttt{1D}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{>=}, \texttt{<=}), \texttt{1D}) \\ &= nmerge(nmerge(\texttt{>=}, \texttt{1D}), \texttt{<=}) \\ &= nmerge(nmerge(\texttt{<=}, \texttt{1D}), \texttt{>=}) \end{aligned} \tag{B.174}$$

$\{\texttt{>=}, \texttt{1O}, \texttt{rO}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{>=}, \texttt{1O}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{>=}, \texttt{rO}), \texttt{1O}) \\ &= nmerge(nmerge(\texttt{1O}, \texttt{rO}), \texttt{>=}) \end{aligned} \tag{B.175}$$

$\{\texttt{>=}, \texttt{1O}, \texttt{1D}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{>=}, \texttt{1O}), \texttt{1D}) \\ &= nmerge(nmerge(\texttt{>=}, \texttt{1D}), \texttt{1O}) \\ &= nmerge(nmerge(\texttt{1O}, \texttt{1D}), \texttt{>=}) \end{aligned} \tag{B.176}$$

$\{\texttt{>=}, \texttt{rO}, \texttt{1D}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{>=}, \texttt{rO}), \texttt{1D}) \\ &= nmerge(nmerge(\texttt{>=}, \texttt{1D}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{rO}, \texttt{1D}), \texttt{>=}) \end{aligned} \tag{B.177}$$

$\{\texttt{<=}, \texttt{1O}, \texttt{rO}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{<=}, \texttt{1O}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{<=}, \texttt{rO}), \texttt{1O}) \\ &= nmerge(nmerge(\texttt{1O}, \texttt{rO}), \texttt{<=}) \end{aligned} \tag{B.178}$$

$\{\texttt{<=}, \texttt{1O}, \texttt{1D}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{<=}, \texttt{1O}), \texttt{1D}) \\ &= nmerge(nmerge(\texttt{<=}, \texttt{1D}), \texttt{1O}) \\ &= nmerge(nmerge(\texttt{1O}, \texttt{1D}), \texttt{<=}) \end{aligned} \tag{B.179}$$

$\{\texttt{<=}, \texttt{rO}, \texttt{1D}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{<=}, \texttt{rO}), \texttt{1D}) \\ &= nmerge(nmerge(\texttt{<=}, \texttt{1D}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{rO}, \texttt{1D}), \texttt{<=}) \end{aligned} \tag{B.180}$$

$\{\texttt{1O}, \texttt{rO}, \texttt{1D}\}$

$$\begin{aligned} \texttt{!=} &= nmerge(nmerge(\texttt{1O}, \texttt{rO}), \texttt{1D}) \\ &= nmerge(nmerge(\texttt{1O}, \texttt{1D}), \texttt{rO}) \\ &= nmerge(nmerge(\texttt{rO}, \texttt{1D}), \texttt{1O}) \end{aligned} \tag{B.181}$$

The statement to be proofed is valid for $n = 3$.

The function definition implies that there is an finite nested formulation for every merge with n operands. For the innermost nesting step the order of operands is arbitrary as shown in the base step. This means that the super ordinate nesting level also has this property. With the result for the underlying nesting level, the proof is also already available through the base step. This is transferable to any nesting level and thus also for any function with n operands.
It has been proofed, that the result of the n-ary merge is independent of the order of the selected operands for the application of the underlying binary merge.