DLR-IB-FT-BS-2023-176

Formal Verification of ML-based Systems in Avionics

Interner Bericht
Hochschulschrift

Autor: Vera Stein

Institutsbericht
DLR-IB-FT-BS-2023-176

Formal Verification of ML-based Systems in Avionics

Vera Stein

Institut für Flugsystemtechnik
Braunschweig

128	Seiten
045	Abbildungen
005	Tabellen
137	Referenzen

Deutsches Zentrum für Luft- und Raumfahrt e.V.
Institut für Flugsystemtechnik
Abteilung Sichere Systeme & SE

Braunschweig, den 14.12.2023

Unterschriften:

Institutsdirektor:	Prof. Dr.-Ing. S. Levedag	[illegible], 15.1.24
Abteilungsleitung:	Andreas Bierig	BW
Betreuer:in:	Alexander Ahlbrecht	A Ahlbrecht Digital signiert von Alexander Ahlbrecht DN: OU=FT-SSY, O=Deutsches Zentrum für Luft- und Raumfahrt, CN=Alexander Ahlbrecht, E= alexander.ahlbrecht@dlr.de Grund: Ich bin der Verfasser dieses Dokuments Ort: Braunschweig Datum: 2023.12.14 10:41:09+01'00' Foxit PDF Reader Version: 2023.2.0
Verfasser:in:	Vera Stein	[illegible]

TU Clausthal

FORMAL VERIFICATION OF ML-BASED SYSTEMS IN AVIONICS

MASTER'S THESIS

presented by

VERA STEIN

Aeronautical Informatics group
Institute of Informatics
Clausthal University of Technology

Vera Stein: *Formal Verification of ML-based systems in Avionics*

STUDENT ID
490919

ASSESSORS

First assessor:	apl. Prof. Dr. Umut Durak
Second assessor:	Prof. Dr. Rüdiger Ehlers
Supervisor:	Alexander Ahlbrecht

DATE OF SUBMISSION
October 20, 2023

EIDESSTATTLICHE VERSICHERUNG

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, wurden als solche kenntlich gemacht. Die Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsstelle vorgelegt.

Clausthal-Zellerfeld, den 20. Oktober 2023

Vera Stein

ZUSTIMMUNG ZUR VERÖFFENTLICHUNG

Ich erkläre mich mit der öffentlichen Bereitstellung meiner Master's Thesis in der Instituts- und/oder Universitätsbibliothek einverstanden.

Clausthal-Zellerfeld, den 20. Oktober 2023

Vera Stein

ABSTRACT

In recent years, many formal methods and tools for verifying Machine Learning (ML) algorithms have been developed. However, they are still in research stage and therefore it is difficult to determine how and to which extent one can use formal verification on a specific Artificial Intelligence (AI)-based system. Due to the lack of guidance for applying formal verification approaches to ML-based systems, they are rarely used in safety-critical applications like avionics which leads to the development of new technologies being restricted in those areas. In order to solve these problems, this work provides a first step for closing this gap by presenting a framework that guides through different verification objectives and supports choosing the right tools for verifying an ML based component. As preparation for the framework, the thesis also includes a systematic summary of state-of-the-art literature on formal verification of ML algorithms. The workflow of the framework will be demonstrated and validated with examples from the avionics domain. The goal of the framework is to contribute to making the integration of ML algorithms in safety-critical applications possible.

ZUSAMMENFASSUNG

Maschinelles Lernen bringt immer mehr Möglichkeiten mit sich und wird somit auch für viele sicherheitskritischen Bereiche wie die Avionik interessant. Damit keine Menschen oder die Umwelt zu Schaden kommen, müssen dort bestimmte Systemeigenschaften garantiert werden. In traditionellen Systemen wird dies of durch formale Verifikation sichergestellt. Diese auf maschinellem Lernen basierenden Systeme gehen aber mit besonderen Herausforderung einher, wie zum Beispiel, dass das System nicht aufgrund von Eigenschaften definiert wird, sondern anhand von großen Datenmengen erlernt wird. Außerdem werden oft große, komplizierte Modelle genutzt.

Zurzeit wird deswegen viel in dem Bereich geforscht, sodass eine Vielzahl an Methoden und Software entstanden ist, die versucht auf unterschiedlichen Modellen verschiedenste Eigenschaften zu verifizieren. Außerdem müssen viele Elemente bei der Verifikation bedacht werden, von den Daten über den Trainingsprozess über das fertig trainierte Modell bis hin zu dessen Implementierung. Insbesondere wenn es eine große Anzahl an Modellen gibt, für die unterschiedlichste Eigenschaften gezeigt werden müssen, wird dies schnell unübersichtlich.

In dieser Arbeit wird ein modellbasiertes Framework vorgeschlagen, um diese komplexen Verifizierungsprozesse und die damit eingehenden Entscheidungen für Eigenschaften und Methoden übersichtlicher und einfach zu machen und damit insbesondere die Verwendung von formaler Verifikation zu ermöglichen. Das Framework unterstützt bei der Definition von Anforderungen an das System, deren Formalisierung in ein oder mehrere verifizierbare Eigenschaften, der Auswahl von für den Anwendungsfall geeigneten Verifikationsmethoden basierend auf Merkmalen des betrachteten Objektes, der formalen Art der zu verifizierenden Eigenschaft und weiteren technischen Merkmalen wie der zur Verfügung stehenden Hardware.

Dafür wurde eine umfangreiche Literaturrecherche nach Definitionen formaler Eigenschaften und Verifikationsmethoden für neuronale Netze durchgeführt. Auf diese Art von Objekten beschränkt sich die detaillierte Implementierung des Frameworks, um den Aufwand der Arbeit im Rahmen zu halten. Die Ergebnisse der Literaturrecherche wurden zu Leitlinien für die Definitionen der Anforderungen und Eigenschaften zusammengefasst. Außerdem wurde eine erweiterbare CSV-Tabelle erstellt, die verschiedene Verifikationmethoden und deren Merkmale beinhaltet. Basierend auf dieser Tabelle und dem in der modellbasierten Softwareentwicklungssoftware Cameo definierten Modell des Systems und der Eigenschaften können dann automatisiert passende Methoden gefunden werden. Weitere Automatisierungen wurden in der erstellten Vorlage implementiert, beispielsweise die automatisierte Ausführung aller zu überprüfenden Verifikationsprobleme und das Zurückführen von deren Ergebnissen auf die ursprünglichen Anforderungen.

Das Framework wurde außerdem auf zwei Anwendungsfälle mit Avionik-Bezug angewendet, um den kompletten Verifikationsprozess zu demonstrieren. Zunächst wird dafür ein in der formalen Verifikation von neuronalen Netzen häufig verwendetes Beispiel namens HCAS verwendet, das durch Aktionsempfehlungen das Zusammenstoßen von Flugzeugen verhindern soll. Es besteht aus 45 relativ einfachen neuronalen Netzen und hier wurde nur das Bestimmen von lokalen Eigenschaften getestet. Aufgrund von den angegebenen technischen Einschränkungen des benutzen Laptops, wurde hier allerdings trotzdem nur ein Tool namens Marabou ausgewählt. Marabou wurde dann auch automatisiert für alle Verifikationsprobleme mit definierten Input-Dateien ausgeführt. Der zweite untersuchte Anwendungsfall ist ein Bedrohungs-Lokalisierungssytem, dass die Koordinaten von anderen Flugzeugen oder Objekten bestimmen soll. Es basiert auf dem YOLOv7-Modell zur Objekterkennung. In diesem Fall gab das Framework keine passenden Verificationsmethoden zurück, weil im Bereich der formalen Verifikation von Objekterkennungsproblem und auch allgemein so komplexen Modellen erst noch Methoden entwickelt werden müssen.

ACKNOWLEDGMENTS

I want to say a big thank you to my supervisor Alex for the guidance through the master thesis process, patiently listening to my progress regularly and giving good remarks. Moreover, I also want to thank the rest of my working group at German Aerospace Center (DLR) especially Jasper, Vincent, Bojan, Wanja, and Sven for some good discussions and technical support and Umut for enabling this thesis, finding an interesting topic, and providing feedback. Further thanks go to my boyfriend Robert, my sister Lena and my parents Angela and Jo for their mental support and grammar corrections.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABKÜRZUNGSVERZEICHNIS

DL	Deep Learning
ML	Machine Learning
AI	Artificial Intelligence
NN	Neural Network
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
NLP	Natural Language Processing
FC-NN	fully-connect feed forward neural network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM-NN	Long Short-Term Memory Neural Network
Res-NN	Residual neural network
ReLU	Rectified Linear Unit
Tanh	hyperbolic tangent
AI-RMF	Artificial Intelligence Risk Management Framework 1.0
ALTAI	Assessment List for Trustworthy Artificial Intelligence
EU	European Union
AIA	Artificial Intelligence Act
CoDANN	Concepts of Design Assurance for Neural Networks
ForMuLA	Formal Methods use for Learning Assurance
MLEAP	Machine Learning Application Approval
MBSE	Model-based System Engineering
SysML	System Modeling Language
BDD	block definition diagrams
ODD	Operational Design Domain
V and V	Validation and Verification
LTL	Linear Temporal Logic
CTL	Computational Three Logic
BMC	Bounded Model Checking
SAT	Satisfiability

MILP	Mixed-Integer Linear Programming
LP	Linear Programming
BaB	Branch-and-Bound
SMT	Satisfiability Modulo Theory
SIP	Symbolic Interval Propagation
PAC	Probably Approximately Correct
LMI	Linear Matrix Inequality
EASA	European Union Aviation Safety Agency
NASA	National Aeronautics and Space Administration
NIST	National Institute of Standards and Technology
VNN-COMP	Verification of Neural Networks Competition
CSV	Comma-seperated values
CPU	Central Processing Unit
GPU	Graphics Processor Unit
DLR	German Aerospace Center
HCAS	Horizontal Collision Avoidance System
COC	Clear Of Conflict
WL	Weak Left
WR	Weak Right
SL	Strong Left
SR	Strong Right
MDP	Markov Decision Process
YOLO	You Only Look Once

1 INTRODUCTION

Because of the great success of ML algorithms, their use is spreading to more and more applications and areas. Beside the growing powers of ML models and algorithms, the increase in available memory and computational power plays a big role in this trend. However, several examples exist [Yua+18] where ML algorithms showed unexpected wrong behavior in certain situations. Due to the black-box behavior of many ML models, it is hard to capture those cases. Especially in safety-critical applications, this hinders the use of such powerful algorithms. Aviation is one of those areas where there is much potential for improvements through ML technologies in safety-critical systems.

Therefore, there has been a lot of research [Zha+; Fer+22; HL20; CH23] going on in the last years that aims to verify properties on such systems and to thereby give guarantees on its behavior. However, the methods of formal verification on traditional hardware and software can not be directly transferred to such systems. Many challenges have to be mastered like the complexity of the models and environments. This also increases the difficulty in defining properties that should be satisfied for a specific application and further general ML-specific properties are developed. However, there exist many different definitions of those properties and the names of them are often not consistent. Moreover, there is a huge amount of verification approaches, methods, and tools with different restrictions, goals, requirements, and subproblems that are addressed. Even though several papers have been published with surveys [CH23; Bri+23] of the tools published in a certain time period that address one kind of property it is still hard to collect all necessary information together. This makes it very hard to apply formal verification techniques to use cases. Some currently emerging guidelines, like the European Union Aviation Safety Agency (EASA) guideline [EASb], are trying to address this problem by defining objectives that should be verified on an AI-based system through its whole life cycle. Still, they lack in giving details on the formal definitions of properties and concrete verification methods. Furthermore, report-based guidelines come with the disadvantage that it is hard to add new methods and tools later and that they cannot be directly integrated into the system.

Therefore, this thesis proposes a model-based framework that guides through the verification of ML-based systems and focuses on formal verification. It thereby addresses the problem of verifying complex systems with many properties. One of the core elements of this framework is a Comma-seperated values (CSV) table that includes state-of-the-art methods and tools for executing the verification with their requirements and restrictions. This table is easily extendable by new tools or characteristics of tools without always having to change the rest of the framework. For concrete guidance on requirements, properties to verify, verification approaches, and tools, the focus will lie on verifying the trained model and in specific Neural Networks (NNs). The verification flow of the framework is also demonstrated for two use cases from avionics: An aircraft collision avoidance system and a threat localization system.

The thesis is structured as follows: Firstly, some preliminaries are introduced. Secondly, the challenges of the verification of ML-based systems are introduced and related laws and guidelines are discussed. Afterwards, the developed framework will be presented. Firstly, this chapter includes a basic overview of the framework and then goes through each step of the verification flow in detail. The next chapter demonstrates how the framework can be applied based on two example use cases. Next, related work is presented and is followed by a discussion on the framework and further ideas that can improve the possibilities of the framework. Finally, a conclusion is drawn.

2 PRELIMINARIES

This chapter includes some background information on verification, machine learning, and model-based system engineering. Those preliminaries are necessary to understand the advantages of formal verification over simulation and are needed to understand the core of this thesis: a model-based framework that supports the formal verification of machine learning based algorithms.

2.1 INTRODUCTION TO VERIFICATION

To find errors and other issues early in the software engineering process the two concepts Validation and Verification (V and V) are used. They support with avoiding unnecessary expenses because the later software issues are found in the life cycle the more expensive it is to solve them. However, in safety critical systems finding software problems too late can even lead to endangering human lives or have a catastrophic impact on the environment. [Spa]

While the goal of validation is to ensure that the specification made about the targeted system really leads to the right product, the goal of verification is to ensure that the built systems satisfies its specification. [AMFM]

As this thesis will focus on the latter, an introduction to different methods of verification will be given in the following sections.

2.1.1 *Verification by Simulation-based Testing*

An intuitive way to test if a system or subsystem fulfills its specification is to define inputs, let the system produce the corresponding outputs, and verify if the outputs comply with the specification. For that, the first big step is to define all realistic inputs. Such an input can be, for example, a list of the values of three different temperature sensors. Since during the verification process no real measured sensor values are taken, but instead possible values are simulated, it is called simulation-based verification. Sometimes, it is more effective to define whole scenarios instead of each single input. Then, it is referred to as scenario-based verification.

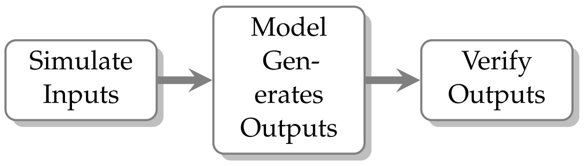


Figure 2.1: Flow of Simulation Verification

2.1.2 *Formal Verification*

Defining all possible inputs and testing them on the model can become a very time-consuming task with increasing input space. Moreover, it often can not be guaranteed that all possible inputs were tested. Those disadvantages of simulation-based testing can be resolved by using formal verification. Formal verification does not look at all the single points in the input space but instead looks at the formal description of the input space and processing on the inputs. This way, the algorithm is verified for all possible inputs which is the main advantage of formal verification.

However, there is also the disadvantage of formal verification that it requires a more mathematical way of thinking in comparison to simulation-based verification which is more intuitive for programmers. In addition, checking properties can sometimes also be really time-consuming. [Gan]

Basically, it is differed between two types of methods for formal verification: Model checking and formal proofs. Model checking consist of formally defining the model and the constraints and then letting a model checking algorithm solve the problem. In contrast, formal proofs are done manually. [Amj]

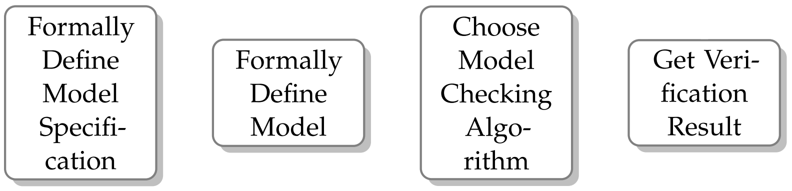

Formally Define Model Specification
Formally Define Model
Choose Model Checking Algorithm
Get Verification Result

Figure 2.2: Flow of Model Checking

2.1.2.1 *Model Checking*

With model checking reactive systems can be examined, i.e. systems that take an input, process it, and return an output and do not initialize execution. The goal of model checking is to verify that a given model ϕ models its specification ψ, also written as

$$\phi \models \psi. \tag{2.1}$$

The model ϕ can be defined as a Kripke structure which is a tuple

$$(S, S_0, T, AP, L) \tag{2.2}$$

with S being a set of states, S_0 the initial states, $T \subseteq S \times S$ being the possible transitions between states, AP being a set of propositions for the labeling and $L : S \rightarrow 2^{AP}$ being the labeling function. In comparison to a Moore or Mealy Automaton it is non-deterministic and in comparison with normal transition systems it has labels. Those two properties make it a good fitting model for model checking. [Sch02; Var09]

The model specification ψ contains all the properties to verify. It can be defined by using Linear Temporal Logic (LTL). The properties can be defined using the standard logical symbols like $\wedge$ (AND), $\vee$ (OR) and $\neg$ (NOT) and additional temporal logic symbols like **G** (Globally), **F** (Eventually), **X** (Next) and **U** (Until) to describe the behavior of the system. Because LTL can only represent properties over all traces of the system, there is the so-called Computational Three Logic (CTL) which is a logic over a tree with branches of infinite length and thereby can model additional properties. [Sch02; Var09]

LTL expressions can also be transformed to non-deterministic Büchi Automatons. Just like finite state machines, Büchi Automatons are a tool to define a language, as the automaton either accepts or rejects a given input word. However, Büchi Automatons have the special property that they can accept or reject words of infinite length. [Sch02; Var09]

To test if the operator $\models$ is satisfied, an automated model checking algorithm is used. For example, a product of the inverse of the model specification in form of a Büchi Automaton and a model in form of a Kripke structure can be calculated. This product can then be examined on emptiness to find out if the model models the specification. If it is empty, there exist no counter example where the model specification is violated and therefore $\phi \models \psi$ is satisfied. (Figure 2.2)

To determine emptiness, the state space is searched with concepts like depth-first search and breadth-first search are applied. Searching through the state space is also referred to as reachability analysis. However, checking explicitly all states leads to the state-explosion problem and therefore is very time-consuming. [Bie+03]

Therefore, besides explicit state model checking, a new type of model checking called symbolic model checking was developed. The idea behind it is to represent sets of states and transitions symbolically. Binary Decision Diagrams can be used to efficiently represent them.

Furthermore, there are SAT-based model checking algorithms. Satisfiability (SAT) based means that they rely on solvers of the Boolean satisfiability problem. Those solvers are able to determine if a satisfying assignment exists for a given Boolean formula. One kind of such algorithms is Bounded Model Checking (BMC) that aims to rapidly find counterexamples. The execution is bounded by some length k and the result is either an counterexample or that there is no counterexample of length k. There exist several other types of SAT-based model checking algorithms like Induction-Based and Interpolation-Based model checking of which details are omitted here. [MRP; Aml+05]

2.1.2.2 *Formal Proof*

The idea behind formal proof verification is that experts manually look at the implementation of the system and create an individual mathematical proof for satisfying certain properties. In contrast to model checking, this process is not automated. Even though this means that the model and the specification do not have to be transformed to a special form, it requires a lot of time of experts to verify one system. [Amj]

2.1.2.3 *Properties to Verify*

With LTL formulas, different properties can be described depending on the verification goal. Often they are in the form that the value of a specific variable has to stay in a certain range.

Another popular verification goal is Logic Equivalence Checking. An example for the need of this is the simplification process of an electrical circuit. It has to be proven that the simplified circuit implements the same logical functions like the complex one, i.e. their logical equivalence has to be checked. [MRP]

2.1.2.4 *Soundness and Completeness*

An important property for algorithms is that they are sound. This means that it only returns that a property is satisfied if it really is. If a verification algorithm is not sound, it cannot guarantee any properties anymore and therefore looses its main advantage over simulation testing.

Another property that verification algorithms strive for but which is less important than soundness is completeness. Complete verification algorithms return always that a property is satisfied or unsatisfied, i.e. they never return that the result is unknown. However, sometimes it is necessary to go without completeness to achieve computational efficiency. [Roc+22]

2.2 MACHINE LEARNING

The main idea behind ML is that a system can learn a behavior from data instead of programming a system with clearly defined rules of behavior. There are three basic paradigms of machine learning: Supervised, unsupervised and reinforcement learning.

In supervised learning, a model is trained on labeled data to learn a relationship between some input variables and one or multiple output variables. Therefore, a dataset is needed for training the model that includes the values of the input variables and the corresponding output for several data points. For example, the input variables of each data point can be the pixels of an image and the output variable can be a class of object that is on the image like cat, dog or airplane. The model is then trained to learn the relation between how an image looks like and what label it has. The input variables are also often referred to as feature variables and the output as target. The probably most well-known type of supervised learning model is the NN which consists of layers of neurons that are connected. The connections represent for example linear relationships combined with non-linearity in form of activation functions.

For unsupervised learning no labeled data is needed. Functionalities like grouping data points into classes or learning representative vectors for words can be learned by an algorithm without supervision. However, it still has to be defined what are good or bad results in the above example respectively by a distance metric or by a second part that reproduces the words from the vectors.

The third type of machine learning is reinforcement learning. Here, an agent is defined that executes an action and gets a reward in each time step. If a neural network is used for determining the agents action than it is also called Deep Reinforcement Learning (DRL). This approach proved to be very useful in many areas as it does not need labeled data but is rather based on exploration instead.

Because the approaches considered later are specialized on NNs, they are introduced in the next section. Afterwards the development process of ML-based systems is regarded. [Nil]

2.2.1 *Neural Networks*

This section will briefly describe different properties of NN like architectures, kinds of layers and activation functions.

2.2.1.1 *Architectures*

There are many different architectures a neural network can have. The simplest one is the fully-connected fully-connect feed forward neural network (FC-NN). Fully-connected means that for each layer all of its neurons are connected to all the neurons of the next layer. The connections between one layer i and the following layer $(i+1)$ consist of a linear operation defined by a weight matrix W and a non-linear transformation defined by an activation function $\sigma(\cdot)$. The values of the neurons of the following layer are represented as a vector z_{i+1} and can be calculated by

$$z_{i+1} = \sigma(W \cdot z_i) \tag{2.3}$$

based on the vector z_i containing the values of the neurons of the regarded layer. The whole model $\hat{f}$ is defined by Equation 2.3 and

$$\hat{f}(x) = \hat{y},\, z_0 = x,\, z_n = \hat{y} \tag{2.4}$$

for an input vector x, a network with n layers, and an output $\hat{y}$. The type of layers which the FC-NN consists of are also sometimes called fully-connected layers or linear layers.

However, depending on the kind of data that is taken as input, other architectures might perform better. For example, Convolutional Neural Networks (CNNs) often perform good on image data. For CNNs, two additional kinds of layers are necessary, convolutional layers and pooling layers. Convolutional layers look at small regions around each pixel to find patterns that can be recognized. Since the same function parameters are used for all regions, convolutional layers can handle big inputs. In addition, the layer fulfills the often desired property for image processing that, for example, an airplane is always recognized as airplane independently of its position in the image. On the other hand, pooling layers sum up a region of pixels to only one output for further processing. For example, this can be done by taking the maximum value of the region, so-called max pooling, or by taking the average of all values in the region, so-called average pooling. This can reduce the input dimension of the next layer and thereby reduce complexity.

For the processing of sequences like in Natural Language Processing (NLP), Recurrent Neural Networks (RNNs) or Long Short-Term Memory Neural Networks (LSTM-NNs) can be used. They have the advantage of saving information about previous outputs in form of an internal state and use it as input for determining the next output.

Furthermore,Residual neural networks (Res-NNs) should be mentioned. They have layers similar to the FC-NNs, however, in them the neurons are not necessarily connected to all of the neurons of the next layer but connections can also skip a few layers. This enables a very efficient training of NNs with a big depth. [RYH22]

2.2.1.2 *Activation Functions*

Activation functions should bring some non-linearities into the model. The most popular ones are the Rectified Linear Unit (ReLU) function, the sigmoid function, and the hyperbolic tangent.

The ReLU function describes the identity for all values greater or equal than zero while it is zero for all negative inputs:

$$\sigma(z) = \begin{cases} z & \text{if } z \geqslant 0 \\ 0 & \text{else .} \end{cases} \tag{2.5}$$

Therefore, by splitting the function at input zero two linear equations can be obtained. This property is used in some verification tools later. [RYH22]

2.2.2 *Machine Learning Process*

The development process of systems with ML-based components differs from the traditional software development process as there is no clear specification of the component but instead data and a model that has to be trained.

After defining the goal of the system, data has to be collected or suitable existing datasets have to be found. At this point already, a lot of aspects have to be considered, for example, data completeness and correctness. Moreover, the data often has to be processed, for example, to get special input features out of raw sensor data. Afterwards, the data has to be split into three parts, one for training, one for validation and one for testing.

Next, the ML model has to be trained. Different hyper-parameter like the architecture, number of layers and neurons, the learning rate, the optimizer and the loss function have to be defined. During training the model learns the weights W. When the model is trained its loss on the validation dataset can be determined and the hyper-parameters can be adjusted, so the model can be trained again for increased performance. Once one decided on the hyper-parameters, the testing dataset can be used to get the performance of the model on unseen data.

There are several concepts for making this training and validation process better like k-fold cross validation, where the model is trained several times on different overlapping parts of the training and based on the results on the fold specific validation data the best model is selected. Such approaches can help, for example, to increase the risks resulting from random sampling and random weight initialization. [Tea]

2.3 MODEL-BASED SYSTEM ENGINEERING

As there often have to be modeled complex systems in avionics which are additionally safety-critical, Model-based System Engineering (MBSE) is often used to keep the overview of all components and functionalities. For this reason and because the verification of ML-based systems can also get complex fast, the MBSE approach is also used for the framework presented later.

The main idea behind MBSE is that all elements that are documented or visualized are connected in one big model. This concept is in contrast to the traditional software engineering where different kinds of documents are in different places designed by different people in different languages. In MBSE the same system elements can be visualized in different diagrams, for example, to show relations to other elements. System components can be shown on a high level view while being linked to their inner specification. [She20] For illustrating the components, connections, and flows the System Modeling Language (SysML) is often used. Typical diagrams for visualization are block definition diagramss (BDDs), package diagrams, and activity diagrams.

A popular tool for MBSE is the commercial tool Cameo from Dassault Systems [Das]. Because it offers many modeling possibilities and the implementation and execution of small supporting scripts called *Opaque Behaviors* it was also used in this thesis. However, the concepts proposed here can also be applied to other tools.

3 CHALLENGES, GUIDELINES, AND LAWS

As AI systems have also application is safety-relevant areas, it would be great to profit of the advantages of formal verification, too. However, using the ideas of formal verification in AI-based systems comes along with some challenges. To still ensure a safe and correctly working system, guidelines and laws concerning AI were developed.

The focus will be on data-driven AI systems, or in other words ML systems, which learn their model from data while model-driven AI is a model developed by experts to behave in an intelligent way.

3.1 CHALLENGES

Several properties of AI systems complicate the formal verification process. Those include problems in formally defining the model specification, the model, its environment, and the properties that should be verified.

3.1.1 *Role of Data*

Since data-driven AI models learn what to output based on huge amounts of data, the specification of the AI is basically the training data. For example, it is not specified that a white object with two wing-formed extensions on a blue background should be detected as airplane. Instead, many pictures that contain airplanes and many pictures that contain something else are used to train the network. Therefore, there is no traditional-like specification for the system what should be detected as an airplane and what not and with that it can not be decided if the model follows its traditional-like specification or not. Thereby, properties that should be verified on the model have to be defined. Further challenges in this area include an automated specification generation from a definition of the desired behavior. [SS16]

Moreover, good data quality has to be assured because the models quality is highly dependent on the data that was used to train it. However, the way in which data is synthetically generated or collected has a big impact on its data quality. There is a need for verifying the selection, design, and augmentation of data. This includes that reasonable inputs were used and that the data is realistic, i.e. is similar to real world behavior. Furthermore, it has to be assured that the data has no bias or variance. Otherwise, unfair behavior could occur, for example, when a pedestrian detection is only trained on white-skinned people and therefore has problems recognizing everyone else. [SS16]

3.1.2 *Model the Learning System*

Further challenges in AI verification are the often really complex architectures of the model and complex learning algorithms. In contrast to traditional hardware and software systems, there is often a high-dimensional input space, for example many pixels as inputs for processing pictures, and an even higher parameter space. In addition, often non-linear activation functions are used that make the model even harder to formally verify.

Moreover, the models parameters cannot be easily interpreted by humans and therefore, a further challenge is to generate explanation for the decisions of the model. To enable this, there is a need for special architectures and the definition of semantic feature spaces instead of low-level view like pixels.

Additionally, the choice of network architectures and learning algorithm is still partly a object of change and try and error. This makes it hard to verify decisions on that.

Furthermore, with AI there is the risk that the trained model contains unknown and unwanted functionalities as it only learns with statistical methods from data. An example of such an unwanted functionality would be that changing only few pixels in a picture leads to completely different categorization while a human cannot see any difference between those pictures. The objective addressing this problem is also called robustness.

Even if most AI systems are deterministic, i.e. the same input leads to the same output, outputs of the model to new inputs are unpredictable and often not intuitive for humans.

In traditional hardware and software systems a system can be decomposed in subsystems which are then verified independently and compositional reasoning can be applied to verify the whole system. However, due to the problem of formally specifying ML models it is even harder to specify them in a consistent way which is required for compositional reasoning. [SS16]

While in hardware design a correct-by-construction approach is used to ensure a systems correctness while constructing it, this is much harder in AI application. Before this challenge can be efficiently addressed approaches for the formal specification of models have to be found. [SS16]

A new challenge that follows from the idea of data-driven learning is the verification of adaptive or so-called online learning algorithms.

They have the special characteristic of constantly learning from the input while being used. This changes the model with every input and therefore model verification would be necessary after each new input.

3.1.3 *Model the Environment*

Because AI enables the interaction with much more complex environments, it comes along with challenges in defining them. While traditional systems model the environment on clearly defined input variables like float values of given temperature sensors and their defined interfaces to the model, in AI systems it often has to be dealed with low-level data like camera or LIDAR data and unknown variables and interfaces. This requires a new way of modeling the environment. [SS16]

Traditional systems could be modeled with automatons and the environment was modeled by non-deterministic transitions. Modeling camera data this way would lead to too many transitions which also leads to the challenge of modeling more complex environments.

Furthermore, probabilistic inputs and outputs of the system and the need of modeling human behavior lead to the challenge of doing environment modeling in a probabilistic and data-driven way.

3.1.4 *Define the verification property and verification outputs*

Moreover, the tasks of ML-based systems are often hard to formalize. Such tasks would, for example, include the detection and classification of objects and natural language processing. Only the end-to-end behavior, i.e. the input and the label can be modeled. [SS16]

While traditional system verification defines logical requirements on safety and liveness and results in a boolean with the value *satisfied* or *unsatisfied*, for ML-based systems also quantitative objective like robustness and performance should be modeled. Verifying those objectives leads to new challenges. Moreover, it lacks on standards methods and metrics for the evaluation. [SS16]

3.2 LAWS

To protect from the risks connected with AI, at least 60 countries have adopted laws and regulation concerning AI. They focus on regulating data in terms of privacy and prohibiting the use of biased data for the training, ensuring safe behavior to prevent harm and defining accountability for decisions. [Gü]

3.2.1 *EU Laws*

The European Commission proposed a set of regulations in 2021 in terms of the Artificial Intelligence Act (AIA). [BUa] The act defines the challenges of AI as its complexity, opacity, unpredictability, autonomy and the role of data. Based on that, they infer the problems of safety risks, fundamental right risks, enforcement, legal uncertainty, mistrust and fragmentation. [Sio]

For addressing those problems they define four categories of applications. Firstly, applications with an unacceptable risk, for example social scoring like it is done in China, secondly high risks applications, for example medical devices or autonomous aviation, thirdly AI with specific transparency obligations, for example bots that do impersonation, and lastly applications with a minimal or no risk. The first category is completely prohibited, for the second and third different requirements and obligations are defined and the last one is permitted with no restrictions. [Sio]

However, there are still some issues in the AIA. Several exceptions are made for using AI for good purposes like finding missing children and those exception rules can be misused. Moreover, the law is not flexible as it focuses on certain applications and no one knows what new areas of AI will emerge in the upcoming years. [BUb]

The AIA is not the only law that influences the use of AI in the EU. Other regulations, like data privacy laws, also include clauses that are at the moment not compatible with most AI systems. One challenge is the demand for explainability which is contained in the European Union's General Data Protection Regulation that says "[automatic processing of data] should be subject to suitable safeguards, which should include [...] the right to [...] obtain an explanation of the decision reached [...]". [HK20]

Furthermore, depending on the application area certification standards are adjusted to also support AI-based systems. For example, in aviation the EASA is already working on guidelines to enable this. They will be further discussed in the next section.

3.2.2 *Further Countries Laws*

Not all countries are working on as strict regulations on AI-based systems as the EU. Regulation can always slow down the process of developing new technologies and applying them to new applications. The UK, for example, published a so-called AI Rulebook which is much less restrictive with the argument that they want to become a *Superpower* in AI technologies. [Gü]

The USA published its regulations on AI in the National AI Initiative and the the Algorithmic accountability act. Their content is rather similar to the content of the EU's AIA. [Gü]

Further countries like China and India are also working on AI specific laws or are including AI-specific clauses in their laws. [Gü]

3.3 GUIDELINES

Several guidelines to address the challenges of AI were developed by national and international organizations and companies. In addition, specific guidelines were developed for different application areas like health, military and aviation. Already for ethical AI guidelines at least 84 documents exist in the world that aim to define them. A selection of the guidelines that were found is presented in the following subsections. [Job]

3.3.1 *SAE International*

SAE International is a global association of engineers and technical experts that is working on mobility solutions. In June 2021 they published a document with the title *Statement of Concerns for Artificial Intelligence in Aeronautical Systems*. [SAE21] The report gives a short overview on the different types of AI algorithms, defines the Machine Learning Life Circle and provides a short analysis of general gaps in the safety of AI. In addition, the gaps in applying the machine learning life-cycle on the different traditional software and Hardware EUROCAE standards were identified.

Moreover, they identify the objectives that should be fulfilled in the different stages of the machine learning life-cycle. In the first stage, they took a look at the *System Definition* where system and safety requirements and objectives are defined and validated. Different hints to consider are given here like the probabilistic nature and unintended behavior of ML systems. The second stage is data selection and validation. A list of data quality objectives are identified there, e.g. correctness, completeness, representativeness, fairness, balance and traceability The next stage consists of the model selection the training and the testing. Different measures were proposed, for example learning strategies that increase generalization like cross-fold-validation, feature selection and random restarts. Further example measures are to ensure performance and functional repeatability and the use of pre-trained models. Additional issues and ideas were proposed for the fourth stages inference implementation like the issue of detecting unexpected features. They also defines some ideas about the verification of the ML sub-system and the system integration, for example, using test data for verification and determining the uselessness of traditional methods.

For the trustworthiness analysis they refer to the European Union (EU)'s Ethics Guideline for trustworthy AI and the seven requirements they define.

3.3.2 *EU's Ethics Guideline for trustworthy AI*

The European Commission published an Ethic Guideline for trustworthy AI in April 2019 [Ind]. According to them, trustworthy AI has to meet three components during its entire life circle, namely being lawful, ethical and robust. The framework which they propose as guidance concentrates on the last two components.

Their proposal of the framework is divided into three steps, starting with the most abstract chapter, namely to discuss the foundations of trustworthy AI. They define four ethical principles that should be adhered to: Respect for human autonomy, prevention of harm, fairness and explicability. Moreover, the second chapter is about requirements and methods for realizing trustworthy AI. Seven different requirements are defined that should be implemented by technical and non-technical methods. The seven requirements include

- ▷ Human agency and oversights
- ▷ Technical robustness and safety
- ▷ Privacy and data governance
- ▷ Transparency
- ▷ Diversity, non-discrimination and fairness
- ▷ Societal and environmental well-being
- ▷ Accountability

The last chapter is about assessing trustworthy AI. There, the resulting trustworthy AI assessment list is defined, which can be used for assessing the trustworthiness of a specific AI-based application. The list includes some assessment questions for each of the requirements they defined in chapter two of their guideline. The list is referred to as Assessment List for Trustworthy Artificial Intelligence (ALTAI) by documents like the EASA AI guidelines.

3.3.3 *EASA AI Guidelines*

The EASA published three papers about AI in the last years. The first document is called *Roadmap 1.0* [EASa] in which challenges of safe AI are defined, AI is classified and a plan for the development of guidelines is presented. The classification is done based on the level of human-interaction of the system. Level 1 are systems where the AI application assists the human. Level 2 are systems where human and machine collaborate and Level 3 are more autonomous machines.

In April 2021 they published their first guideline on Level 1 application [EASc] and in February 2023 they published an extended and edited version which handles Level 1 and Level 2 applications safety [EASb]. The following paragraph will refer to the more recent version. It should also be mentioned that the guideline is limited to supervised offline AI-systems.

Their framework for creating a trustworthy AI system consists of four building blocks. The first one is AI trustworthiness analysis which is based on the seven requirements defined in the EU's Ethics Guideline and on the questions from the ALTAI adapted to aviation. Answering those questions should help with characterizing the application and doing a safety, security and ethics-based assessment.

The second block they define is called AI Assurance and consists of two parts: Learning assurance and development explainability. Learning assurance addresses new assurance challenges that come with the integration of learning algorithms in software systems. The part about development explainability suggests ideas to handle the problem of understanding models that were learned by machines.

The third block is called Human Factors for AI. Different issues that can appear in the human-machine interaction are considered in that block. This includes operational explainability, i.e. that the machine can explain its decisions when cooperating or collaborating with a human. Another important point of this block are considerations about human-AI teaming.

The last block is dealing with AI Safety Risk Mitigation. This includes the implementation of additional safety measures, to avoid hazard in case something is going unexpectedly wrong.

For each of the four building blocks a set of objectives was defined. These objectives are already detailed and thereby give guidance of what to consider to create a trustworthy AI-based system.

3.3.4 *NIST*

The National Institute of Standards and Technology (NIST) of the USA published an Artificial Intelligence Risk Management Framework 1.0 (AI-RMF) in January 2023 [NIS01] which goals were directed by the Nation AI Initiative Act. Adherence to the framework is voluntary but provides some guidance for companies on how to develop safe AI systems.

In the first part of the document they discuss the risks that arise with the integration of AI in systems and define characteristics for an trustworthy AI, namely:

- ▷ Valid and reliable
- ▷ Safe
- ▷ Secure and resilient
- ▷ Accountable and transparent
- ▷ Explainable and interpretable
- ▷ Privacy enhanced
- ▷ Fair

The second part of their document describes the guidance on developing trustworthy AI based on four functions: Govern, map, measure, and manage. The idea behind the *govern* function is to manage everything and hold the other components together on a high level. Examples of subcategories of the govern function are the management legal requirements, making sure that the characteristics of a trustworthy AI are used, and taking care of the risk management process. The *map* function is for defining and understanding the context and impacts, categorization of the system, and mapping the risks to the system components. The goal of the *measure* function is to analyze, access, and monitor risks. This also includes to evaluate which trustworthiness characteristics the system implements. The *management* function aims to measure risks regularly, prioritize them, implement strategies for maximizing benefits, and manage also the risks and benefits of third-party entities.

The document also proposes to use the framework with so-called profiles that are defined for different use-cases like hiring and fair housing. With the report, a playbook was published to give some intuition on how to navigate and use the framework.

3.3.5 NASA

The National Aeronautics and Space Administration (NASA) published in April 2021 a document titled *NASA Framework for the Ethical Use of Artificial Intelligence* [NAS04]. They consider to differ between today's simple Narrow Intelligence (ANI) and future human-level Artificial General Intelligence (AGI) and Artificial Super Intelligence (ASI). Additionally, like the EASA they differ between different human interaction levels.

The core of their paper is the definition of six ethical AI principles:

- ▷ Fair
- ▷ Explainable and transparent

▷ Accountable

▷ Secure and safe

▷ Human-centric and societally beneficial

▷ Scientifically and technically robust

Their principles are based on the ones developed by the Department of Defense's Defense Innovation Board, Gartner, and the American Council for Technology Industry Advisory Council. They map these principles to their applications and they add the lastly mentioned principle to ensure that the AI-based system is consistent with their specific scientific NASA methods.

3.3.6 *Companies AI Guidelines*

It should also be mentioned that big IT companies like Google and Microsoft defined their own guidelines for AI. For example, Google published its seven objectives for AI applications, including being socially beneficial, avoiding unfair bias, and being accountable [Goo]. Microsoft also published a guideline [KFG] including 18 tasks that should be done categorized by when they have to be done in the development process. However, as their goal is not to restrict their technologies, they rather focus on where they can add an additional step on defining or correcting something.

3.4 VERIFICATION THROUGH SYSTEM DEVELOPMENT

Many of those guidelines address the problem of defining what objectives have to be met during the different verification steps that have to be taken during the development of an ML model are illustrated in Figure 3.1.

The verification of the ML learning model starts when the data that should be used for the training is collected or chosen from external sources. The raw data is often preprocessed afterwards and new features are calculated from the given ones. Several aspects have to be taken into account that have to be checked on the preprocessed data and can be optionally already checked on the raw data, for example, it has to be ensured that the data is representative, complete, correct, and fair.

Further, when deciding on the model's architecture and the learning process it should be checked that none of the requirements is violated by the decisions. For example, for some kinds of explainability specially designed models are necessary.

After the model is trained, further requirements have to be checked on the trained model. For example, it has to be ensured that it is robust and satisfies certain safety properties.

Later in the development process, the ML model is implemented on the hardware it should operate on and integrated into the whole system. This is called the inference model and it has to be verified that no new errors arise during this implementation process.

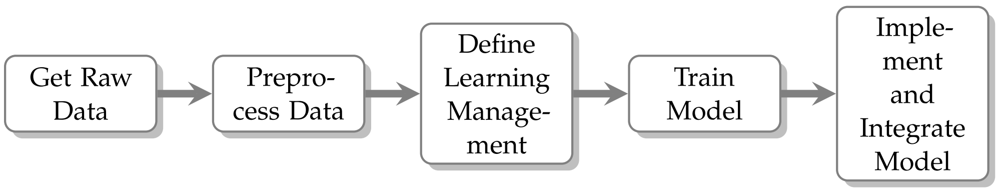


Figure 3.1: Steps of ML Development

In this thesis, the focus will lie on the verification of the trained model and the other elements are only included in the work in general.

4 DEVELOPMENT OF A CONCEPTIONAL FRAMEWORK

The main contribution of this thesis is the development of a conceptual framework that supports the formal verification of systems with ML-based components. The framework will be presented in this chapter by first introducing the general idea, structure, and verification process. Afterwards, the elements of the framework and the verification steps are regarded in detail. For the detailed requirement, property, and verification method selection process only trained NNs are considered as objects to limit the scope of this work.

4.1 GOAL AND APPROACH

Many different tools are developed for the formal verification of ML-based components. However, they are often only applicable for specific models, specific properties, or restricted by other constraints. Therefore, if a user wants to verify his specific ML-based system, it is hard to figure out what methods and tools would work for him. Moreover, the verification of a complete ML-based system requires verifying many different properties on different objects. This comes with the additional challenge of defining useful properties and keeping the overview over many verification tasks.

The goal of this framework is to support the user with these challenges and those presented in section 3.1. A model-based approach is chosen to handle complex systems with many components and requirements that should be verified. The tool Cameo [Das] is used in this implementation of the framework as a basis for modeling. As the training data is what mainly defines an ML-based component and no detailed system description is designed that can be checked, additional ML-specific requirements like generalization and robustness and their formal definitions are collected from the literature and grouped. Moreover, a literature review of verification approaches and tools for the different properties of NNs was done and the process of finding fitting methods was automated. In addition, because still many new methods and tools are proposed, a format for an easily expandable table was proposed. Besides appending new lines for new tools or methods, also new technical requirements or properties of objects can be added as columns. Moreover, consideration were made on how to represent ML models like NNs and other components needed in the model-based environment to still have all necessary information for modeling the verification process.

4.2 STRUCTURE AND BASIC ELEMENTS

The basic packages that have to be defined for completely verifying a system are further grouped here into three packages: *Requirements and Properties to Verify*, *System Definition and Learning Management*, and *Verification*. Furthermore, the framework consists of a *Language* package where the frameworks' definitions of stereotypes and enums are defined. The structure of the whole framework is shown in Figure 4.1 and the three main packages will be further described in the following subsections.

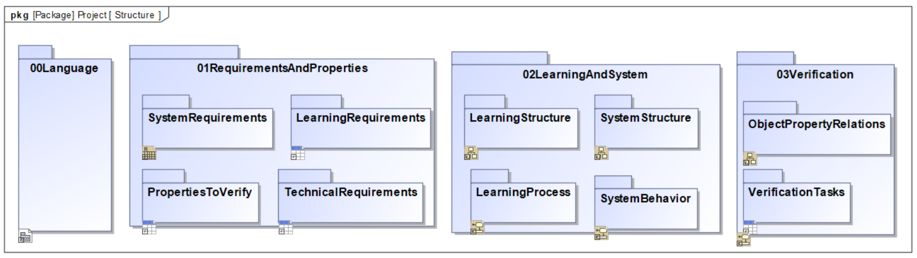


Figure 4.1: Verification Structure

4.2.1 *Requirements and Properties to Verify*

Besides the basic category of requirements that can be used as abstract requirements for hierarchical refinement later, there are three kinds of requirements defined in the framework, namely *system requirements*, *learning requirements*, and *technical requirements* as shown in Figure 4.2.

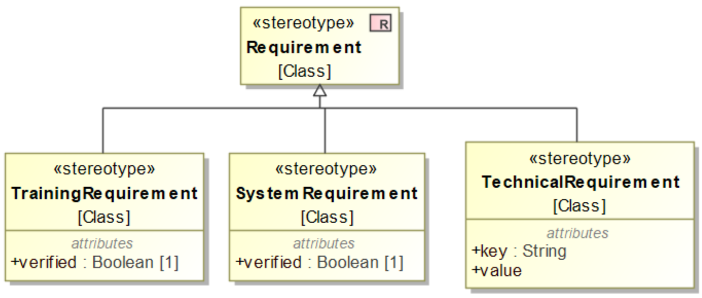


Figure 4.2: Language Specification of Requirements

The system requirements specify what the end system should fulfill. A requirements table is provided in the *SystemRequirements* package in which all of them can be listed. The system requirements that should be checked later are on the lowest level of nested structures while basic category requirements can be used to group them.

Learning requirements define constraints on elements of the learning process like on the data or on the algorithm that is used for the training. The end-system is indirectly dependent on these elements and therefore, they should be part of the verification process, too. However, their detailed definition is out of the scope of this thesis. They can also be listed in a table.

Technical requirements are constraints on the training and verification process. This includes, for example, verification solver licenses and hardware constraints like a Graphics Processor Unit (GPU) being required. They are listed in a table of the package *TechnicalRequirements*.

After the training and system requirements are defined they have to be formalized to one or more formal constraints which are called *properties to verify* in the language specification as shown in Figure 4.3. This stereotype inherits from the standard stereotype *ConstraintBlock*.

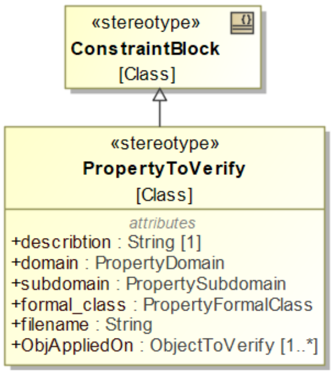

«stereotype»
ConstraintBlock
[Class]
«stereotype»
PropertyToVerify
[Class]
attributes
+describtion : String [1]
+domain : PropertyDomain
+subdomain : PropertySubdomain
+formal_class : PropertyFormalClass
+filename : String
+ObjAppliedOn : ObjectToVerify [1..*]

Figure 4.3: Language Specification of Properties to verify

They are connected to the system requirements by so-called *refine* relations. When a property to verify is created then its attributes have to be defined too. Besides a textual description and a formal definition a *domain*, a *subdomain*, and a *formal class* have to be specified. These values can be selected from enums defined in Figure 4.4.

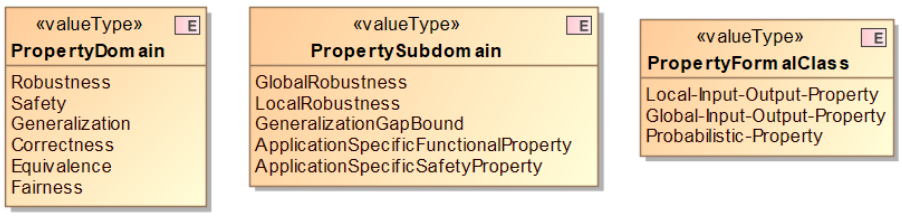


Figure 4.4: Enums of the attributes of *PropertyToVerify*

A further attribute of the properties to verify is *ObjAppliedOn* which defines for which objects the property should hold. For example, if a system consists of several NNs for different use cases a property might only be defined for a few NNs. Based on this, the script *generateRelationsPropsToObj()* can be used later to generate *ToProofOn* dependencies between properties and objects.

4.2.2 *System Definition and Learning Management*

This package includes diagrams defining the elements and activities of the learning process and the structure and behavior of the developed system. To define the structure of the learning process and the system structure a Block Definition Diagram (BDD) is used for each of them. Activity diagrams are used to define the learning activities and the behavior of the system, however, they are until now not needed for the verification. The BDDs contain all the elements on which verification can be performed having the stereotype *ObjToVerify* or at child stereotype of it and can furthermore include relations between them or further elements. For example, if there are different datasets that are used to train different NNs, a connection between each data object and the corresponding NN can be created. This might be interesting for special properties that have to be defined for a model and its training data. An example are data- and model dependend generalization properties. There are more specific kinds of the stereotype *ObjToVerify* like the *ML_Model* for all kinds of ML models, the *DataObj* for data object, and the *Neural_Network* being a specific stereotype for NNs. The stereotypes needed in this work and some others to demonstrate further possibilities are shown in Figure 4.5. However, this is only a small amount of specific stereotypes that can be defined here in the future. The corresponding enums can be found in Figure 4.6

4.2.3 *Verification*

After the objects of examination, the properties to verify, and the technical requirements are defined, a fitting verification method has to be determined and the verification has to be executed. The Verification package includes all components for this verification process. This includes the opaque behaviors that are small scripts to support with the verification steps and the elements that contain the information on the verification tasks and their results.

The opaque behavior *DetermineMethods()* supports this by checking which methods are compatible and returns a list of them. Moreover, it generates the *VerificationTask* objects for each single verification

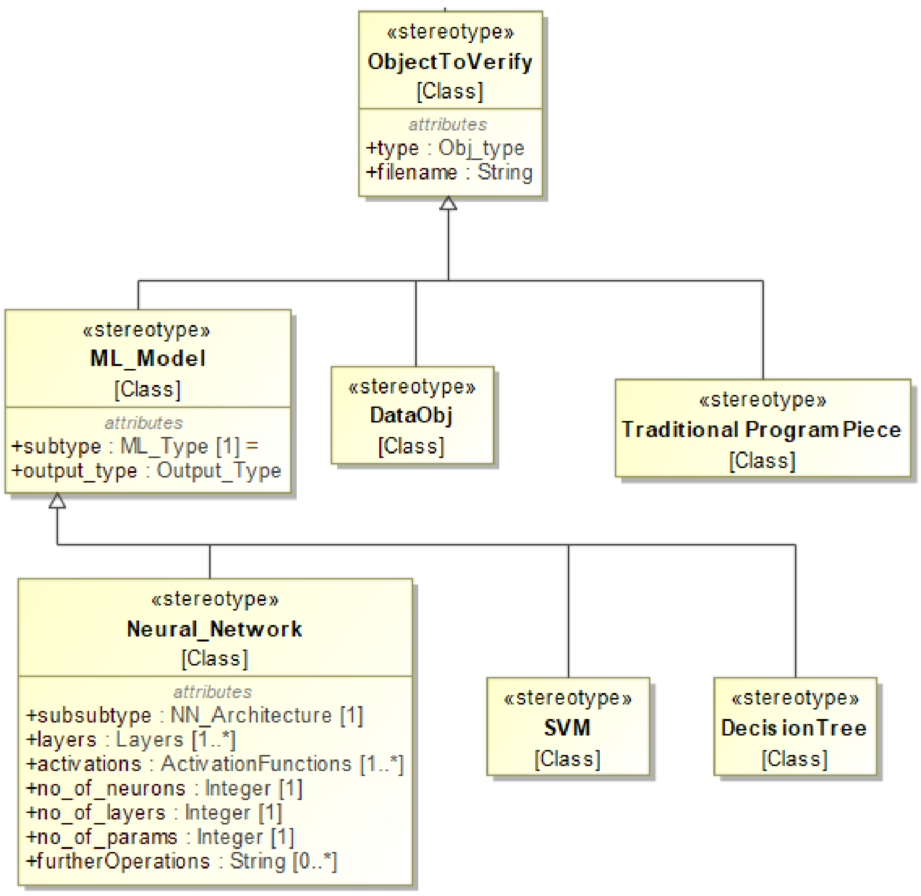


Figure 4.5: Language Specification of Objects to verify

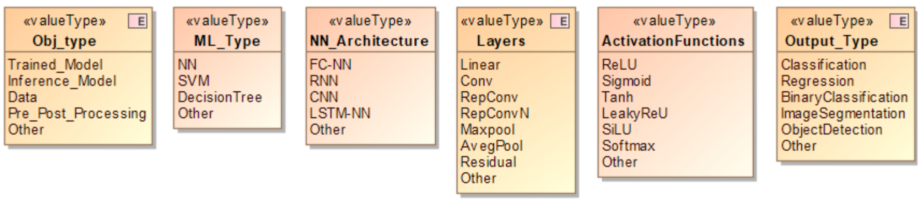


Figure 4.6: Language Specification of Objects to verify

problem. As shown in Figure 4.7 they save the object that is regarded, the property that should be verified on the object, and the generated list of matching verification methods.

The method selection process will be presented later in more detail. It is based on a CSV table that includes the important state-of-the-art verification methods with their properties. However, the table can be easily extended by new methods or other properties.

An overview of all verification tasks created for an use case can then be found in the *VTasks* table. The verification methods can be executed by the opaque behavior *ExecuteVerification()*. In future work,

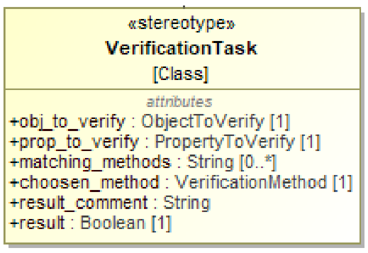

«stereotype»
VerificationTask
[Class]
attributes
+obj_to_verify : ObjectToVerify [1]
+prop_to_verify : PropertyToVerify [1]
+matching_methods : String [0..*]
+choosen_method : VerificationMethod [1]
+result_comment : String
+result : Boolean [1]

Figure 4.7: Language Specification of Verification Tasks

the execution of all methods can be automated, however, until now only one tool is in the list of installed methods. Thereby, the complete flow can be seen on some example properties.

Since there can be a huge number of verification tasks, the result if the verification was successful or not should also be mapped to the requirement for seeing which of the requirements are already satisfied. This is done with further opaque behavior called *MapResultsOfVtasksOnReqs()* that automatically checks if all verification tasks connected to the requirement had positive results and then sets the requirements attribute *verified* to true. Otherwise the attribute verified is set to false.

4.3 VERIFICATION FLOW

The verification flow is illustrated in Figure 4.8 and contains only the steps of the system development that are necessary for verification. First, the learning and system requirements, the learning management and system structure including the objects that should be checked, and the technical requirements are defined by the user with some basic guidance. Then, the learning and system requirements are refined into properties to verify and their attributes are defined. Both steps are done by the user with the support of the guidelines. Moreover, *Refine* dependencies are created between learning or system requirements and properties to verify by the user. Afterwards, the user has to start the opaque behavior *generateRelationsPropsToObj()* to connect properties and objects based on the attribute *ObjsAppliedOn* of the property to verify by *ToProofOn* dependencies.

Next, the user has to start the opaque behavior *DetermineMethods()* that creates *VerificationTask* objects for each of those dependencies and determines fitting methods. Out of the list of fitting verification, one has to be chosen by the user. Then the verification tasks that have an installed method selected and an object and property file defined can be executed automatically. Afterwards, the opaque behavior

MapResultsOfVtasksOnReqs() can be started to create the possibility of later regarding the verification results of each requirement.

4.4 GUIDANCE ON THE DEFINITION OF REQUIREMENTS

All the guidelines mentioned in section 3.3 described a similar set of requirements for the development of trustworthy AI systems. The EASA even gave a broad list of detailed objectives for the different life cycle stages of such systems. Many of these objectives deal with defining and documenting different parts and being aware of special properties of the system. This thesis will focus on formal verification and will only cover the details on requirements and properties on the trained model. Therefore, in the next step, it will be analyzed which of the objectives found in guidelines contain some form of system requirements to verify. Afterwards, examples of technical requirements on the verification process will be provided.

4.4.1 *System Requirements*

Based on existing guidelines and the definition of ML-specific properties in the literature on verification, six main objectives were defined. The system requirements can be derived from these categories.

The first kind of verification objective is to verify the application-specific requirements of the model that was trained. This is also defined in the EASA guideline as LM-10. The satisfaction of the same application-specific requirements must also be verified later on the inference model, the model that is implemented on the end-product hardware and software. This is called objective IMP-09 in the EASA guideline. [EASb]

To distinguish clearly between safety-relevant requirements and other use case specific requirements on the system, the application-specific requirements are split into objectives here: Safety and correctness.

Safety requirements define that specific 'bad' states are never reached. An example of a safety requirement in an air collision avoidance system would be that if an intruder is close and on the left side of the own-ship, the own-ship is not advised to drive in that direction.

Correctness requirements can be functional requirements of the system, or efficiency requirements, for example, that an aircraft should not do unnecessary maneuvers if there is no danger, or requirements that ensure consistency with the physical laws. For example, if aircraft intruders are supposed to be localized based on camera data, it can be

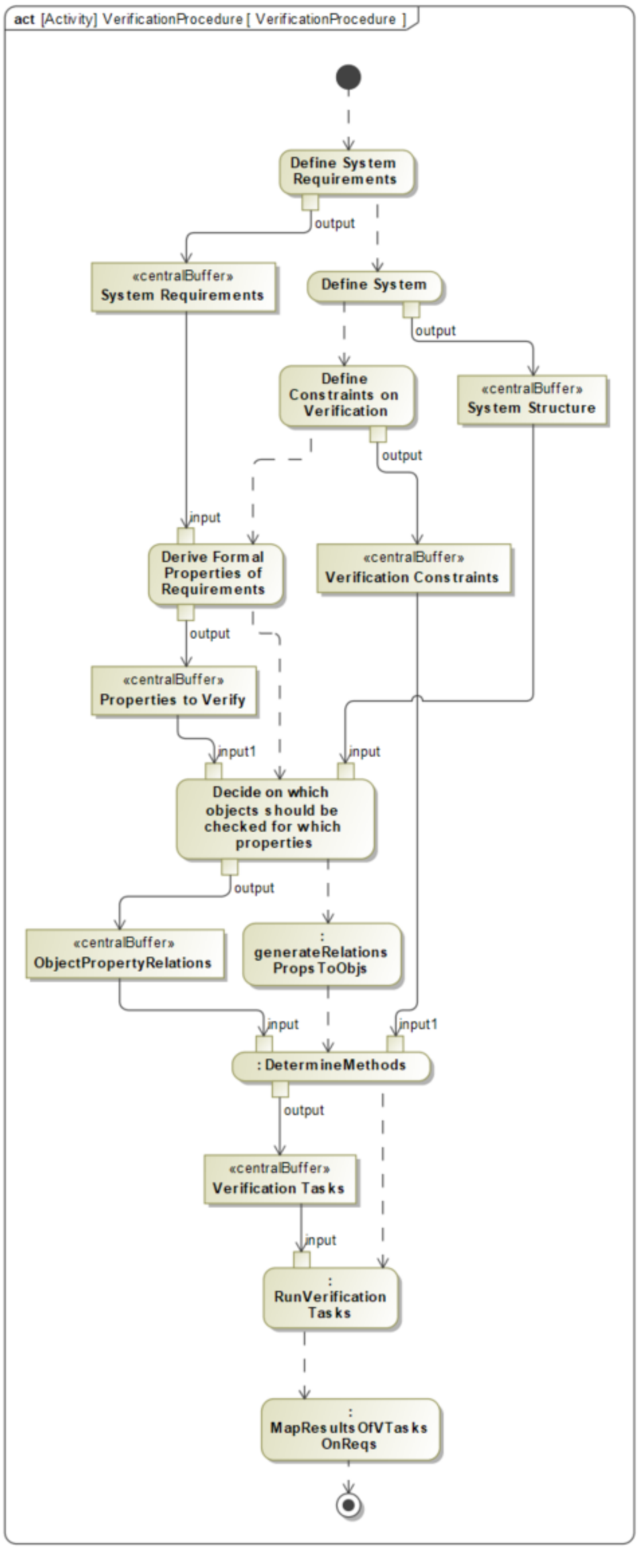

act [Activity] VerificationProcedure [VerificationProcedure]
Define System Requirements
output
«centralBuffer» System Requirements
Define System
output
Define Constraints on Verification
«centralBuffer» System Structure
output
input
Derive Formal Properties of Requirements
«centralBuffer» Verification Constraints
output
«centralBuffer» Properties to Verify
input1
input
Decide on which objects should be checked for which properties
output
«centralBuffer» ObjectPropertyRelations
: generateRelations PropsToObjs
input
input1
: DetermineMethods
output
«centralBuffer» Verification Tasks
input
: RunVerification Tasks
: MapResultsOfVTasks OnReqs

Figure 4.8: Verification Process

encoded that intruder 1 can not be closer than intruder 2 if intruder 2 is closer than intruder 3 and intruder 3 is closer than intruder 1. [Alb21]

The third objective that should be considered is robustness. Verifying that an AI-based system is robust is really important as it was shown that changing only a few pixels of a picture can lead to completely different classification results while humans can see no difference. [Hua+20b]

Such examples, in which a little bit of noise in the input leads to a completely different and wrong output, are also called adversarial examples. Besides evoking safety issues, they can also be used target-oriented by cyber-attackers to threaten the security of the system, for example, by finding adversarial examples in malware detection or by putting some small targeted stickers on a stop sign so that an autonomous car cannot detect it anymore.

The EASA guideline also mentions that robustness should be verified on the training model in objective LM-13 and later on the inference model as mentioned in objective IMP-08. [EASb]

Another ML-specific objective is generalization. Having trained a model on a big dataset and getting a good performance on that data does not mean that the performance will also be good on unseen data. As this is a common problem for Deep Neural Networks (DNNs), there is a need for some generalization guarantees, generalization verification, or other measures addressing the issue.

The EASA guidelines define two objectives for this, one to ensure that a specified generalization bound is satisfied (EASA LM-14) and one to provide some generalization guarantees (LM-04). [EASb]

Depending on the application, further objectives like fairness and equivalence might also be important. As there is also research ongoing on formally defining such properties and developing methods for verifying them, they will also be considered here.

To sum up, the step of defining the system requirements consists of taking a closer look at the six main objectives shown in Figure 4.9 - Functional Correctness, Safety, Robustness, Generalization, Fairness, and Equivalence - and how they can be applied for the specific use case. Based on that, the user can create the system requirements.

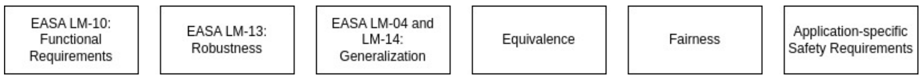

EASA LM-10: Functional Requirements
EASA LM-13: Robustness
EASA LM-04 and LM-14: Generalization
Equivalence
Fairness
Application-specific Safety Requirements

Figure 4.9: Main Objectives

4.4.2 *Technical Requirements*

Several restrictions might be made for the execution of the verification method depending on the technical conditions or requirements on the verification algorithm. Based on the requirements needed by the tools from literature presented in section 4.8 the following list of technical verification requirements was created but can be extended further depending on which tools will be developed in the future:

- ▷ Needed hardware (e.g. GPU, CPU)
- ▷ Needed licenses (e.g. Gurobi)
- ▷ Need for completeness
- ▷ Time constraints (e.g. verifies a specific benchmark in less than 60 seconds)
- ▷ Data formats one can provide the objects and properties in
- ▷ Operating system to run the tool on

Many tools use parallel computing in their algorithms to get faster results. This is often done using CUDA [Nvi] from Nvidia wherefore their algorithms only work on Nvidia GPUs. When it is mentioned that a GPU is needed, it often particularly means a Nvidia GPU. Moreover, some tools encode the handled problems as optimization problems and use a solver like Gurobi [TBB] to solve the problem. Such solvers may need licenses. Furthermore, not all verification algorithms considered here are complete as completeness comes with compromises on efficiency. In some cases, where a complete algorithm is needed this property can be formulated in the technical verification requirements. In addition, there are slower and faster tools, so measuring how fast a tool is on some specific benchmarks can also be of interest. Constraints on that could also be defined in the list of technical verification requirements. Moreover, different input formats are used by the tools. Depending on the formats the user can provide the inputs in, tools with other input formats can be removed from the list of fitting methods. Not all tools can be installed on all operating system. This should also be captured in the definition of the technical requirements.

Not all of technical requirements mentioned above have to be considered. If the user of the framework does not care about the operating system because he can install any operating system then he does not have to mention it in the list. In contrast, if it is required that no further license is need, the user has to explicitly mention it in the table by setting *licenses=[]*.

4.5 DEFINITIONS OF VERIFICATION PROPERTIES IN THE LITERATURE

While the last section regarded what main verification objectives there are for which requirements can be defined, this section will take a look at how the requirements can be formalized to verification properties. This is based on literature research and the found definitions were grouped according to the objectives. However, many words like correctness and robustness are not used consistently in the literature but have several meanings. Therefore, this section should provide names for the definitions that are used in the rest of the framework.

4.5.1 *Correctness*

Correctness requirements might have any form of content as they are completely application-specific. Therefore, it is really hard to define a formalization mechanism. However, the requirements considered for formal verification are often restricted to those that can be converted to Input-Output properties, i.e. constraints on the input can be defined that should lead to some constraints on the output.

Formally, a precondition ϕ on the inputs $x \in \mathcal{X}$ from the input space $\mathcal{X}$ and a so-called post-condition ψ on the outputs $\hat{y} = f(x)$ are defined. The function $f(x)$ is the function that is implemented by the regarded neural network. For all inputs x that fulfill ϕ their corresponding neural network output should fulfill ψ, i.e.

$$\forall x \in \mathcal{X} : x \models \phi \Rightarrow \hat{y} = f(x) \models \psi. \tag{4.1}$$

[Bri+23]

To verify other correctness requirements simulation-based verification can to be used. This is also what the EASA AI guideline proposes to handle application specific requirements. They suggest setting up test cases to see if the requirements are fulfilled on them. In addition, it should be made sure that the test cases cover all relevant inputs. However, as mentioned in subsection 2.1.2 an advantage of formal verification is that it can prove that a property holds for all inputs of the input space. Different formal methods for requirement-based verification were gathered in section 4.8. [EASb]

4.5.2 *Safety*

As the safety requirements are also application-specific the same possibilities for formalization exist that were already mentioned in the subsection about correctness. Several methods to support determining risks and hazards for use cases in software engineering have been proposed. However, they are their own research area and therefore, this thesis assumes that risks and risky situations were already determined.

4.5.3 *Robustness*

Robustness is an important objective for ML-based systems. Many different definitions and approaches exist for addressing its verification. In the EASA guideline, they suggest adding some noise to the original inputs and seeing how the model performs on them. However, that is just one approach. Before formal verification approaches will be considered in the next chapter, the objective of being robust should be defined. [EASb]

In classification, the idea of robustness and stability is that for small changes in the input, the predicted class does not change. In regression, the predicted output should not deviate too much when there is a small difference in the inputs. The verification objectives robustness and stability are getting a lot of attention in the literature in formal verification methods for ML algorithms. Nevertheless, their definitions strongly vary between research papers and are sometimes mixed up with each other definitions. Therefore, the different kinds of robustness and stability will be explained and assigned names for later reference.

4.5.3.1 *Local Stability*

Most often by robustness *local stability* is meant. The word *local* indicates that the stability property is regarded around one input x_0 while the word *stability* indicates that only inputs x with noise are considered that are still in the Operational Design Domain (ODD).

Local stability can be formalized as an input-output property which was described in the previous section. For that, the precondition ϕ is set to $||x - x_0||_p \leqslant \delta$ with some norm p and maximum distance δ. Therefore, only inputs are considered that have a distance of at least δ to x_0. In classification, the output class predicted for x should then still be the same class as the one of x_0. The true target value of x_0 is known because it was taken from the labeled dataset. The whole property of local stability for classification problems can be formulated as

$$\forall x \in X_{ODD} : x \in \{x : ||x - x_0||_p \leqslant \delta\} \Rightarrow \hat{y} = f(x) \in \{\hat{y} : \hat{y} == f(x_0)\}. \quad (4.2)$$

However, as $f(x)$ is often assumed to not hold the class itself but the probabilities for the different classes, the local stability is also sometimes formulated by

$$\forall x \in X_{ODD} : x \in \{x : ||x - x_0||_p \leqslant \delta\} \Rightarrow \hat{y} = f(x) \in \{\hat{y} : \hat{y}_{i^*} > \hat{y}_j, \forall j \neq i^*\}$$

(4.3)

with j being the index of the classes and i^* being index of the class with the highest probability in $y = f(x_0)$.

In case of regression, a maximum error ϵ between the outputs has to be defined. The main idea is that if the input x is close to x_0 then the output $f(x)$ should also be close to $f(x_0)$ or formally written

$$\forall x \in X_{ODD} : x \in \{x : ||x - x_0||_p \leqslant \delta\} \Rightarrow \hat{y} = f(x) \in \{\hat{y} : ||\hat{y} - f(x_0)||_p < \epsilon\}. \tag{4.4}$$

In order to define norm p, different kinds of norms were used in the literature including the Manhattan norm L_1 , Euclidean norm L_2 and Maximum norm L_∞ which are defined by

$$L_1 : \qquad ||x||_1 = \sum_{i=1}^{n} |x_i| \tag{4.5}$$

$$L_2 : \qquad ||x||_2 = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{4.6}$$

$$L_\infty : \qquad ||x||_\infty = max(x_1, x_2, ..., x_n) \tag{4.7}$$

for a vector x of length n. A further norm that is used to define the distance between two vectors x and x' is the L_0 norm which is defined by

$$L_0 : \qquad ||x - x'||_0 = \sum_{i=1} \mathbf{1}_{\{x_i \neq x'_i\}} \tag{4.8}$$

For example, for a picture as input x and a noised picture as input x' it counts the number of pixels that are different in the pictures. [Hua+20b; CW17; Men+22]

However, it should be remarked that these norms defining how similar two inputs are do not always agree with what a human would perceive as similar.

4.5.3.2 *Distinct Local Stability*

In Equation 4.3 the assumption was made that the network outputs probabilities for the different classes of the classification problem. Based on this a threshold θ can be defined that specifies how big the probability of the highest class $c(x)$ for an input x should be at least. Similar to Equation 4.3 the *distinct local stability* is defined as

$$\forall x \in X_{ODD} : x \in \{x : ||x - x_0||_p \leqslant \delta\} \Rightarrow \hat{y} = f(x) \in \{\hat{y} : (\hat{y}_{i^*} > \hat{y}_j, \forall j \neq i^*) \vee (c(x) < \theta)\} \tag{4.9}$$

[LK22]

4.5.3.3 *Global Stability and Lipschitz Constants*

Sometimes when looking at regression problems, it is required that the output does not change too suddenly between all similar inputs, instead of only examining the local areas around some labeled data points. The therefore specified *global stability* is defined for arbitrary inputs x and x' by

$$\forall x, x' \in X_{ODD} : x \in \{x : ||x - x'||_p \leqslant \delta\} \Rightarrow \hat{y} = f(x) \in \{\hat{y} : ||\hat{y} - f(x')||_p < \epsilon\}. \tag{4.10}$$

This property can be transformed into the mathematical property of being Lipschitz continuous with the Lipschitz constant $l = \frac{\delta}{\epsilon}$ by the following transformations on the constraints in Equation 4.10

$$||x - x'||_p \leqslant \delta \Rightarrow ||f(x) - f(x')||_p < \epsilon \tag{4.11}$$
$$\Longrightarrow \neg(||x - x'||_p \leqslant \delta) \vee ||f(x) - f(x')||_p < \epsilon \tag{4.12}$$
$$\Longrightarrow \neg(||x - x'||_p \leqslant \delta \wedge ||f(x) - f(x')||_p \geqslant \epsilon) \tag{4.13}$$
$$\Longrightarrow \neg(\epsilon \cdot ||x - x'||_p \leqslant \delta \cdot ||f(x) - f(x')||_p) \tag{4.14}$$
$$\Longrightarrow ||f(x) - f(x')||_p \leqslant \frac{\delta}{\epsilon} \cdot ||x - x'||_p. \tag{4.15}$$

Consequently, this constraint restricts how much the output can deviate in relation to their inputs. The factor, that compensates that input and output space may scale differently, is the Lipschitz constant $l = \frac{\delta}{\epsilon}$. Consequently, a neural network is globally l-stable if and only if the function f that it implements is Lipschitz continuous with the constant l. [KS23; Zha+22a]

4.5.3.4 *Local Robustness*

The difference between *stability* and *robustness* is that a stable system only has to be able to handle noisy inputs within the normal ODD while a robust system should be able to react correctly to external influences and outsider data. Because of this, it is not only defined for all $x \in X_{ODD}$ but for all inputs in the input space X_{All}. Similar to the local stability the *local robustness* can be defined as an input-output-property for classification as

$$\forall x \in X_{All} : x \in \{x : ||x - x_0||_p \leqslant \delta\} \Rightarrow \hat{y} = f(x) \in \{\hat{y} : \hat{y} == f(x_0)\} \tag{4.16}$$

and for regression as

$$\forall x \in X_{All} : x \in \{x : ||x - x_0||_p \leqslant \delta\} \Rightarrow \hat{y} = f(x) \in \{\hat{y} : ||\hat{y} - f(x_0)||_p \leqslant \epsilon\}. \tag{4.17}$$

However, as local input points are regarded this focuses on looking at edge cases. [EA23]

4.5.3.5 *Global Robustness*

Similar to global stability, *global robustness* can be defined as

$$\forall x, x' \in X_{All} : x \in \{x : \|x - x'\|_p \leqslant \delta\} \Rightarrow \hat{y} = f(x) \in \{\hat{y} : \|\hat{y} - f(x')\|_p \leqslant \epsilon\}. \tag{4.18}$$

Checking the robustness and not only the stability of the system can be very important especially if the used sensor data is sometimes faulty.

4.5.3.6 *Probabilistic Stability and Robustness*

Another emerging kind of robustness is *probabilistic robustness*. It follows the approach of not requiring the robustness property to hold for all inputs x but only for most inputs by working with probability theory.

Different attempts were made to reach this goal. Webb, Rainforth, Teh, and Kumar [Web+19] proposes to measure how robust a system is based on the probability that the examined property is not fulfilled. They express that the property is not satisfied for an input x as $s(x) \geqslant 0$ and define the distribution of the input space of interest by $p(x)$. They define the probability of failure as

$$P_{X \sim p(\cdot)}(s(X) \geqslant 0) = \int_X \mathbb{1}_{\{s(x) \geqslant 0\}} p(x) dx \tag{4.19}$$

with $\mathbb{1}_{\text{Condition}}$ being the indicator function. They look at the challenge of determining this probability and use it as a measure for robustness. [Web+19]

Mangal, Nori, and Orso [MNO19] defines that a neural network that satisfies probabilistic robustness is robust with $(1 - \epsilon)$ probability. While in the previous definition of probabilistic robustness an arbitrary property $s(x) \geqslant 0$ was used, this definition uses a similar definition of robustness to local and global stability and robustness. A network is defined by them to be probabilistic robust for a given k, δ and ϵ if

$$P_{x, x' \sim p(x)}(\|f(x') - f(x)\| \leqslant k \cdot \|x' - x\| \mid \|x' - x\| \leqslant \delta) \geqslant 1 - \epsilon. \tag{4.20}$$

The method RoMA [LK22] provides a definition that is a mixture of the distinct local stability (Equation 4.9) and the probabilistic robustness (Equation 4.20). They end up with the following expression for the distinct probabilistic stability measure:

$$P_X(\operatorname{argmax}(f(x)) == \operatorname{argmax}(f(x_0))) \vee (c(x) < \theta) \mid \|x - x_0\|_\infty \leqslant \delta). \tag{4.21}$$

4.5.3.7 *Lyapunov Stability*

In the word *Lyapunov stability* the word *stability* has a rather different meaning than in the previous definition. Lyapunov stability is defined for dynamical systems with an equilibrium state is for example used in physics for modeling the behavior of a pendulum and in engineering for modeling the behavior of a quadrotor. However, nowadays these dynamics are sometimes also approximated by neural networks. Then, it can be examined if such a network satisfies the Lyapunov stability which describes that the values get close to the equilibrium in each time step. A neural network function f, that calculates the next state $x_{t+1} = f(x_t, u_t)$ based on the previous state x_t and the controller u_t for a time step t, is stable if there exists a Lyapunov function V with the following properties

$$V(x_t) > 0 \qquad \forall x_t \neq x^* \tag{4.22}$$

$$V(x_{t+1}) - V(x_t) \leqslant -\epsilon V(x_t) \qquad \forall x_t \neq x^* \tag{4.23}$$

$$V(x^*) = 0 \tag{4.24}$$

for an equilibrium state x^* and a scalar $\epsilon > 0$. [Dai+21]

4.5.3.8 *Further Definitions of Stability and Robustness for Neural Networks*

Further kinds of stability and robustness are defined in the literature like *noise sensibility* [Aro+18]. Additional robustness definitions are based on how many of specially generated adversarial examples failed [CW17]. The latter also leads to interesting results, however, it only offers an upper bound for the robustness and does not give any guarantees. Another definition is the one of targeted robustness proposed by Gopinath, Katz, Pasăreănu, and Barrett [Gop+18b] which only allows inputs to be labeled as long as they are not wrongly getting one special class as the label.

4.5.4 *Generalization*

An ML model generalizes well if its performance on unseen data is similar to the performance on the train data. This is an important verification objective as ML models sometimes tend to overfit. Overfitting is when an algorithm rather learns the training data with its labels by heart instead of learning the relationship between inputs and outputs. This might be the result of too many parameters in the learning algorithm, however, having too few parameters leads to underfitting, i.e. the learned model is much too simple and does not capture the function.

The EASA also addressed this problem of assessing how well a model is generalizing and how generalization abilities can be approved in its

AI guidelines, the Concepts of Design Assurance for Neural Networks (CoDANN), and the Machine Learning Application Approval (MLEAP) project. They suggest to determine how well an algorithm generalizes based on the so-called generalization gap. The gap should measure the differences between the loss on the training dataset and the loss there would be on the underlying distribution $\mathcal{D}$, formally

$$G(\hat{f}) = |\mathbb{E}_{D\sim\mathcal{D}}[l(\hat{f}, D)] - L(\hat{f}, \mathcal{D}_{train})| \tag{4.25}$$

for a model $\hat{f}$, $l(\hat{f}, D)$ being the loss function, and $L(\hat{f}, \mathcal{D}_{train})$ being the averaged loss on the training data. Since the underlying distribution is unknown in most cases, this definition is not practical, and upper bounds of the generalization gap have to be assessed. There are different approaches to assessing it.

The generalization gap can be estimated by taking into account the test data. The expected loss on all data $\mathbb{E}_{D\sim\mathcal{D}}[l(\hat{f}, D)]$ can be approximated by the loss of the evaluation of the model on the test data $\mathcal{D}_{test}$, i.e. data that was not used during the training of the model. Formally, one obtains

$$G_{test}(\hat{f}) = |L(\hat{f}, \mathcal{D}_{test}) - L(\hat{f}, \mathcal{D}_{train})|. \tag{4.26}$$

[EASb] Other metrics $D(.)$ than subtracting and taking the absolute can also be used like the Euclidean distance. Furthermore, the loss function can also be replaced, for example, by taking the performance function instead or by replacing it with the expected value $\mathbb{E}$ of the loss L.

The generalization gap has to be small enough to say that a model is generalizing well. Therefore, it has to be compared to some value δ which can also be dependent on other parameters, for example, the number of data points and model complexity measurements.

However, on the true gap only probabilistic statements can be made. Therefore, it is common to model this by Probably Approximately Correct (PAC) based methods. As the name describes they are of the form

$$\mathbb{P}[G(\hat{f}) < \delta(.)] > 1 - \epsilon \tag{4.27}$$

with ϵ being the parameter for defining the *Probably* and $\delta(.)$ for defining the *Approximately*.

One possibility is to choose $\delta(.)$ by considering the model's complexity without regarding the dataset. There exist several methods based on the Vapnik–Chervonenkis (VC) dimension d_{VC} for determining how complex relations a model can express. This is only dependent on the model, not on the data. An example of such a model complexity d based bound is

$$\forall \epsilon : \mathbb{P}[G(\hat{f}) < \frac{\ln(d) + \ln(\frac{1}{\epsilon})}{m}] > 1 - \epsilon \tag{4.28}$$

with m being the dimension of the dataset. [BM21; VPL20]

Neural networks have many parameters and thereby a high complexity d. They are sometimes trained on a lower number of data points m than their amount of parameters. This leads to a high value of $\delta(m, d, \epsilon)$. However, NNs have been shown to generalize well in practice. Hence, using this definition for NNs often leads to really loose bounds. [Shi; EAS]

Therefore, approaches that also depend on the training data were developed. One example is the Rademacher Complexity Bound which depends on the training error and the margin. It leads to tighter bounds than the VC-based bound, however, the bounds are often still too loose. [Shi; VPL20]

Different kinds of PAC-Bayes bounds are promising approaches developed in the last years. As the term *Bayes* suggests, a prior and posterior distribution are considered. The prior P describes the model's distribution before being trained and the posterior Q is the distribution after being trained. The distance from Q to P is measured based on the Kullback-Leibler divergence KL(.). The exact definitions of δ vary from bound to bound. An example is the McAllester bound which is defined as

$$\forall \epsilon : \mathbb{P}\left[G(\hat{f}) < \sqrt{\frac{KL(Q||P) + \frac{5}{2}\ln(m) + \ln(\frac{1}{\epsilon}) + 8}{2m - 1}} \right] > 1 - \epsilon. \quad (4.29)$$

[Alq23; VPL20; VPL20; EAS; Bel+23]

[Bel+23] provides a draft of a list with possibilities for how to calculate generalization bounds. For neural networks PAC-based bounds are also listed there.

Furthermore, in the field of generalization considerations on the distributions of the training and test dataset are made. Here, it can be distinguished between taking the test dataset from the same distribution as the training dataset or from out-of-distribution. As an example, an autonomous vehicle can be trained on scenarios in New York and then be tested on different scenarios in New York. However, it can also be trained on data from New York and be tested on data from the small town of Clausthal. Hence, for the generalization gap, one would probably get completely different results in the two cases. [Kir+23]

In some papers like [FV19] the generalization ability is also defined by uniform stability. Uniform stability describes how stable the performance of a model is if the training data is changed. This can, for example, be examined by removing one data point from the training dataset.

There exist several further exciting ideas on how to define generalization like from [Zha+21] on how the trained network compares to a network trained with randomly labeled data. Amir, Maayan, Zelazny, et al. [Ami+23] proposes an approach based on Karenina's hypothesis. In a nutshell, this hypothesis states that if many randomly initialized models are trained those that are close to each other are probably also close to the true functions. So the model where the most other models are closed generalizes the best. A further idea is based on model compression [Che+20] where the model is compressed to a smaller model to get the tighter bounds of the compressed model. In addition, Ju, Li, and Zhang [JLZ23] proposes an approach based on the Hessian matrix. However, as not all of them can be discussed here, it is referred to the literature. A good survey on how good the different kinds of generalization measure really work was published by [Jia+19].

4.5.5 *Further Verification Objectives*

While the three objectives, generalization, robustness, and application-specific input-output properties are the most important things that one wants to verify on the trained network, there are still some further objectives that might be interesting in some cases. So they will be quickly presented.

4.5.5.1 *Equivalence*

As neural networks can get huge, there is effort in compressing them. However, afterward, it must be verified that they still implement the same, or almost the same, function on our ODD as before. Given that there are two NNs that implement the functions f and g respectively, perfect equivalence between them can be expressed as

$$\forall x \in \mathcal{X}_{\mathcal{ODD}} : f(x) = g(x). \tag{4.30}$$

Nonetheless, perfect equivalence is often not needed and not realistic in the case of compression. Therefore, [KBKS20] defined the notation of ϵ-equivalence as

$$\forall x \in \mathcal{X}_{\mathcal{ODD}} : ||f(x) - g(x)||_p < \epsilon \tag{4.31}$$

for some norm p to express that each output of the first NN on an input should be really close to the output of the other NN on the same input. This definition is especially useful for regression tasks.

For classification, they propose the term top-1-equivalence which defines that the NNs should output the same class as the class with the highest probability for a given input. For that, $\hat{y} = f(x)$ and $\hat{y}' = g(x)$ are set and

$$\forall x \in \mathcal{X}_{\mathcal{ODD}} : \mathrm{argmax}_i(\hat{y}_i) = \mathrm{argmax}_j(\hat{y}'_j). \tag{4.32}$$

[KBKS20; Teu+21]

4.5.5.2 *Fairness*

Another big topic in NN verification is fairness as in the past several cases were found where algorithms learned racist or sexist behavior. Given a dataset with a set of attributes $\mathcal{A}$ as inputs, it first has to be defined which of them should not influence the decision of the algorithm. These so-called protected attributes $\mathcal{P}$ can include, for example, race and sex for an algorithm that predicts the salary.

Based on that three different definitions of fairness are given by [BR22]. The first one is *individual fairness* which specifies that no two data points should lead to different results if they just differ in protected attributes. Formally, there exist no two data points x and x' with

$$(i)\forall j \in \mathcal{A} \setminus \mathcal{P} \qquad x_j = x'_j \text{ and} \tag{4.33}$$

$$(ii)\exists k \in \mathcal{P} \qquad x_k \neq x'_k \text{ and} \tag{4.34}$$

$$(iii) f(x) \neq f(x'). \tag{4.35}$$

However, sometimes the requirement (i) is too strong and therefore is changed to only restricting that the values of all non-protected attributes are close. This is also called *ϵ-fairness* in the literature and defined as, there exist no two data points x and x' with

$$(i)\forall j \in \mathcal{A}\ \mathcal{P} \qquad |x_j - x'_j| \leqslant \epsilon \text{ and} \tag{4.36}$$

$$(ii)\exists k \in \mathcal{P} \qquad x_k \neq x'_k \text{ and} \tag{4.37}$$

$$(iii) f(x) \neq f(x'). \tag{4.38}$$

The third definition of fairness they provide is *targeted fairness* which adds additional application-specific constraints on the input features. In general, the fairness verification splits into two parts, depending on if the neural network is assumed to be a black-box or a white-box. While black-box verification can only look at different inputs and their corresponding outputs, white-box verification aims to look at causes for unfair behavior within the decision structure of the NN. A verification algorithm that outputs several adversarial examples can already help with understanding the source of the unfair behavior. [BR22]

A weaker variant of the individual fairness is *group fairness* which is, for example, mentioned by [Fel+15] and [Urb+20]. It measures if there is some discrimination between two different groups instead of comparing individual data pairs on sensitive properties. One kind of group fairness is the demographic parity property which is, for example, defined by [BZSL19] for binary classification as

$$\frac{\mu_{R_{min}}}{\mu_{R_{maj}}} \geqslant 1 - \epsilon \tag{4.39}$$

for a given $\epsilon \in [0, 1]$. In group fairness a minority and a majority group are compared. The trained model f can get the values from the minority group as input X_{min} or can get inputs from the majority group X_{maj}. The variable $\mu_{R_{min}} = \mathbb{E}[f(V_{min})]$ is the expected value of the output on the data of the minority group and the value $\mu_{R_{maj}} = \mathbb{E}[f(V_{maj})]$ of the majority group. Therefore, the definition formalizes that the minority group has not much smaller chances for a positive result $f(x) = 1$ than the majority group.

However, there are a huge number of further fairness variants and subvariants. A good summary of fairness properties was recently given by [CH23]. They gather the different methods for estimating and enhancing fairness on the data before training, during training, and on the trained model. Moreover, they group them regarding their approach and outline that most approaches to measuring fairness are only defined for binary classification tasks. However, transferring these definitions to regression tasks is becoming a topic in fairness verification, too.

4.5.5.3 *Security*

The security aspects of the trained model are partly covered by the previously defined objects like robustness against adversarial examples. Further thoughts regarding security have to be made on the whole system and its implementation, however, they are out of the scope of this thesis. [EASb]

4.6 guidance on choosing verification properties

The following guidelines should help to orient oneself in the huge number of definitions for verification properties that were presented in section 4.5. If different definitions should be applied for different models each property has to be defined on its own. For example, for a system with two neural networks, NN1 and NN2, NN1 might only be verified to be locally stable while NN2 has to be globally stable.

4.6.1 *Robustness*

Table 4.1 shows an overview of the decisions that have to be made when selecting a type of robustness. All decisions can be combined with each other in a reasonable way but not for all combinations there already exist verification methods.

Firstly, it has to be decided if stability or general robustness is needed. They differ in the size of the input space for which the robustness property should hold. Often it is enough to take a look at all inputs from the ODD and therefore decide in favor of stability. Secondly, robustness can hold locally or globally. If it is enough that the property holds locally around a finite number of input points, local robustness can be chosen as it is much simpler and faster. In addition, if it is only important that the property is fulfilled most of the time and not always, the adjective 'probabilistic' can be added to the robustness definition name. Moreover, for some classification problems, the class with the highest probability is only further processed if its probability is higher than a threshold. In this case, the distinction property can already be added in the robustness verification property.

Weaker Property	**Stronger Property**	**When to choose which**
Stability	General Robustness	Stability if it is sufficient to ensure robustness for inputs of the ODD. General Robustness else.
Local	Global	If it is enough to proof robustness only locally around specific data points or around all points in the training data, local Robustness. Else Global Robustness.
Probabilistic	Non-Probabilistic	Probabilistic, if the application can tolerate a few counterexamples of the property to verify. Else Non-Probabilistic.
Distinct	Non-Distinct	Use distinct robustness for classification problems where the post processing is depending on the probability of the highest class being higher than a threshold.

Table 4.1: Properties of the Robustness Definition

The type of the robustness property can be chosen from Figure 4.12.

Furthermore, for most robustness definitions a norm has to be chosen. It depends on the data and the application which L_p - norm is the best. In cases where the number of different inputs counts for measuring the perturbation, the L_0-norm can be used. For image data as input, the L_∞-norm turned out to be close to the similarity people see in images. However, sometimes the L_1- and L_2-norm distance measurement is useful, too. [Men+22; CW17]

The values for the parameters ϵ and δ have to be chosen based on the use case. For example, in an image classification case, where the pixels have gray-scale values between zero and one, δ can be chosen as 0.01, and as norm the L_∞ norm can be selected. So the property that will be verified is that the image is still classified to the same class if all pixels are at most one percent brighter or darker.

4.6.2 *Generalization*

In subsection 4.5.4 it was discussed that the most popular way of measuring the generalization ability of an ML model is by estimating the generalization gap through an upper bound. Many formulas for generalization bounds exist, however, the PAC-Bayes-based bounds often lead to tighter bounds for NNs than those that only measure the complexity of the model without considering the data. Table 4.2 shows a short overview.

Property	**When to use**
Model-based Probabilistic Generalization	▷ Rather simple bound only based on the complexity of the model ▷ Can be very loose, especially for models with many parameters like NNs
Model- and Data-based Probabilistic Generalization	▷ Based on model complexity and indirectly on the training data ▷ More complex calculation ▷ Often tighter bounds for NNs
Train- and Test-data based Loss	▷ Simple to calculate ▷ Does not tell anything on the true generalization gap ▷ Highly relies on the selection of the training and testing data

Table 4.2: Generalization Properties

4.6.3 *Correctness*

The correctness properties should formally represent the system requirements and are thereby very use-case specific. If possible, they should be formalized in the format of local input-output properties as for this class the most general tools exist. Global Input-Output properties are also possible as shown in Table 4.3.

Property	**When to use**
Local Input-Output Property	▷ Use if possible ▷ Model can have black-box behavior ▷ Many tools for verifying it exist
Global Input-Output Property	▷ Stronger than Local Input-Output Properties ▷ Use for properties that contain constraints on the relation between any two points in the input space ▷ Harder to verify
Probabilistic Property	▷ If only Probabilistic statements can be made ▷ Not many tools for general probabilistic properties exist yet

Table 4.3: Correctness Properties

4.6.4 *Safety*

The safety properties depend on the individual risks of the use case. These risks were analyzed when the system requirements were defined. Like the correctness requirements, they should be formalized to local input-output properties if possible.

4.6.5 *Fairness and Equivalence*

Fairness and equivalence are objectives that do not necessarily play a role in all use cases.

Fairness properties only have to be defined if the system or its environment has sensitive features, for example, age, sex, and ethnicity. Many definitions for fairness properties exist and for use cases with fairness as a high priority, the definition that fits the best to the use case has to be searched in the literature. The paper of Caton and Haas [CH23] is a good starting point for that. As there are so many definitions of fairness, this thesis will only focus on the two most important classes, individual fairness, and group fairness. Some characteristics are listed in Table 4.4 to get some guidance when regarding a specific use case.

Fairness class	**When to use**
Group Fairness	▷ If only two groups have to be distinguished ▷ Looks only at the probabilities for a positive result in the groups ▷ If only between-group issues are relevant ▷ Weaker than individual fairness
Individual Fairness	▷ If fairness definition can be based on selected sensitive features ▷ Verify that there is no case where only sensitive features are different and the output is different ▷ Does not address that other non-sensitive feature might correlate with sensitive features

Table 4.4: Main Fairness Classes

Equivalence properties are only relevant if there is an ML model in the system that has to be equivalent or almost equivalent to some other ML model. Table 4.5 summarizes the characteristics of different equivalence properties.

Equivalence class	**When to use**
Perfect Equivalence	▷ Regression problems ▷ If it is important that for all inputs from the ODD the output is exactly the same ▷ Seldom useful in practise
ϵ-Equivalence	▷ Regression problems ▷ If the outputs of te models only need to be similar ▷ More practical as ϵ can be choosen depending on the use case
1-top-Equivalence	▷ Classification problems ▷ If only the class with the best output is relevant or if the models are then close enough
Probabilistic Equivalence Properties	▷ Weaker than non-probabilistic ▷ If probabilistic guarantees on the equivalence are enough

Table 4.5: Equivalence Properties

4.7 FURTHER STEPS FOR THE VERIFICATION PROPERTIES

By deciding on the objective and the kind of property, one only defines the *domain* and *subdomain* of the *PropertyToVerify* object. The other attributes that were shown in Figure 4.3 still have to be defined. The next two attributes that will be examined more closely are *formal_class* and *ObjAppliedOn*. Further attributes would be the *description* which is simply a string describing the property textually for an overview, and the *filename* which should later include the formal definition of the property in the right format for being used as input for a fitting verification tool.

4.7.1 *Formal Classes for the Verification Properties*

Before existing approaches are regarded, the objective should be categorized in different categories depending on their formal definition. Depending on this, different approaches can be used later.

Next, all objectives are considered in more detail with their types of formal properties and the formal class they can belong to.

As correctness and safety properties are completely use-case specific, the three types of properties defined for them were already distinguished by their formal class. As shown in Figure 4.10 and Figure 4.11 they can be sorted into the three formal classes *local input-output properties*, *global input-output properties*, and *probabilistic properties*. Further kinds of properties might also be defined in the future that do not belong to any of the three classes.

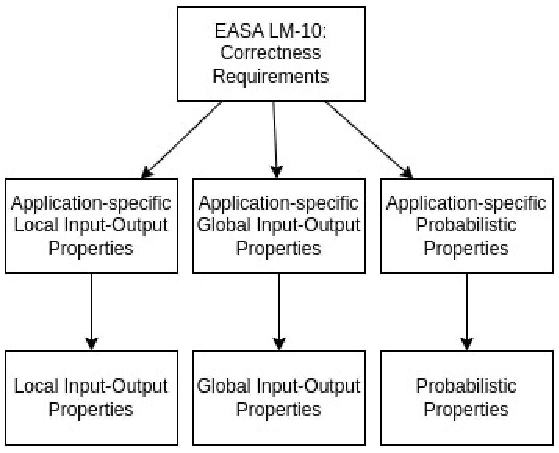

EASA LM-10: Correctness Requirements
Application-specific Local Input-Output Properties
Application-specific Global Input-Output Properties
Application-specific Probabilistic Properties
Local Input-Output Properties
Global Input-Output Properties
Probabilistic Properties

Figure 4.10: Considered Types of Correctness

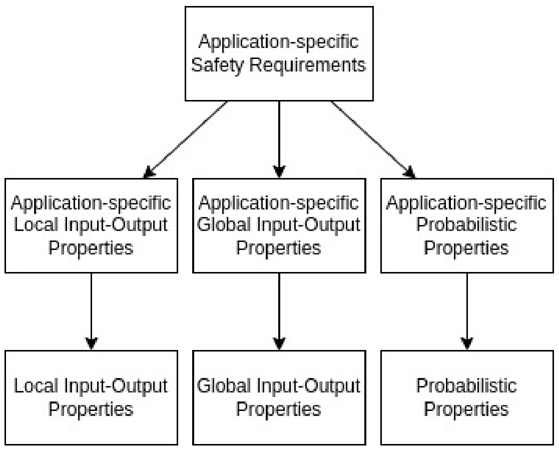

Application-specific Safety Requirements
Application-specific Local Input-Output Properties
Application-specific Global Input-Output Properties
Application-specific Probabilistic Properties
Local Input-Output Properties
Global Input-Output Properties
Probabilistic Properties

Figure 4.11: Considered Types of Safety

The robustness properties were defined more specifically in subsection 4.5.3 and subsection 4.6.1 and are sorted to the same four formal classes according to their formal definitions like shown in Figure 4.12.

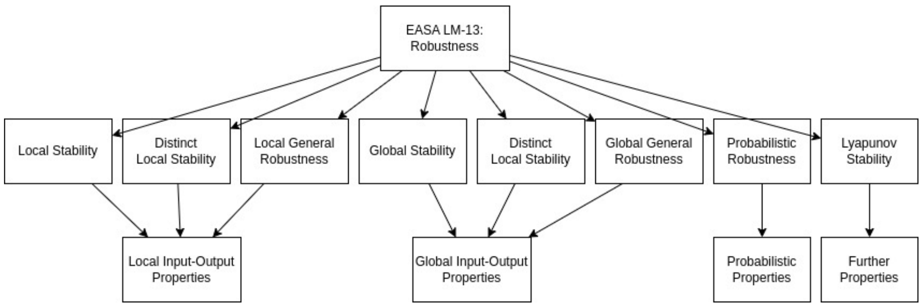


Figure 4.12: Considered Types of Robustness

As the generalization gap is unknown if the true data distribution is unknown, only probabilistic guarantees can be given on it. Another possibility is to use non-formal testing-based methods. Figure 4.13 shows the generalization property type grouping.

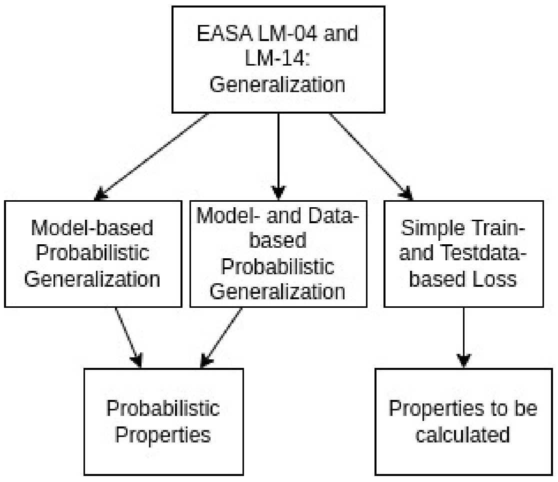

EASA LM-04 and LM-14: Generalization
Model-based Probabilistic Generalization
Model- and Data-based Probabilistic Generalization
Simple Train- and Testdata-based Loss
Probabilistic Properties
Properties to be calculated

Figure 4.13: Considered Types of Generalization

Fairness properties can be formulated as global input-output properties or as probabilistic properties as shown in Figure 4.14. Generalization properties compare the outputs of two models and thereby cannot be handled by local input-output properties which act on one input and one output or global input-output properties which work on two inputs and one output. Therefore, they were sorted into the class *further properties* which would need a closer look in future work. Figure 4.15 shows their formal classification.

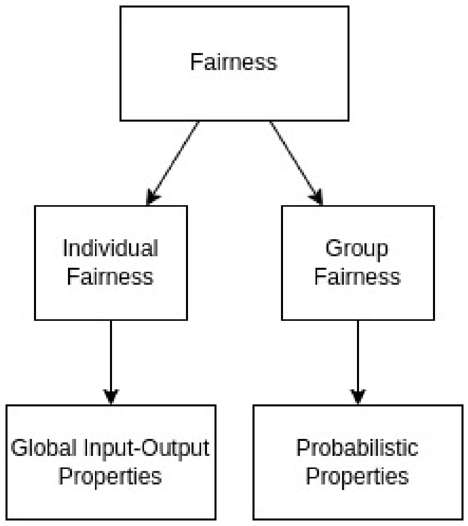


Figure 4.14: Considered Types of Fairness

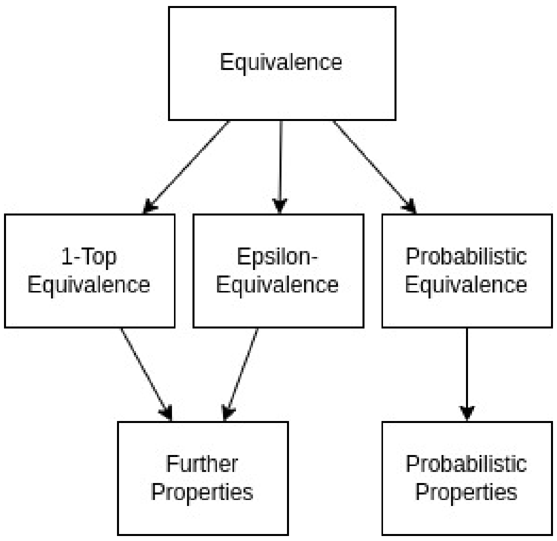


Figure 4.15: Considered Types of Equivalence

4.7.2 *Select Objects They Refer To*

The properties might only belong to one object but might also have to be verified on several objects. Therefore, a list of objects has to be given in the *ObjAppliedOn* attribute of the property to verify. Moreover, the opaque behavior *CreateObjPropRelations()* has to be executed. It uses the defined list to create *toProofOn* relations between the properties and each object of the *ObjAppliedOn* list.

4.8 APPROACHES AND TOOLS FOR FORMAL VERIFICATION

This section provides a literature review on what methods and tools there are for verifying the properties defined in the last steps. The next steps of the framework can be found in the section afterwards.

4.8.1 *Approaches for the Verification of Local Input-Output Properties*

Local input-output properties have great importance in the formal verification of NNs as they include local stability, local robustness, monotonicity, and application-specific local safety and correctness properties. Many methods for verifying this type of properties have been proposed in the last years. There are three basic approaches that were used or combined with other techniques: Reachability analysis, optimization, and search. [Liu+20]

Those approaches will be discussed in this section to gain a basic understanding of the theory behind the specific tools that will be discussed in the next subsection.

4.8.1.1 *Reachability*

The idea behind reachability analysis is to look at the outputs that can be produced by the model with some specified input space. Those outputs are also referred to as the reachability set which is formally defined as

$$\mathcal{R}(\mathcal{X}_\phi, f) := \{y : y = f(x), \forall x \in \mathcal{X}_\prec\} \tag{4.40}$$

with $\mathcal{X}_\phi = \{x \in \mathcal{X} : \phi\}$ including all points of the input space that fulfill the input constraint ϕ. The reachability set $\mathcal{R}$ can be calculated layer by layer and the way in which they are calculated differs between the tools.

The original input-output property verification problem was described in Equation 4.1. Using the sets $\mathcal{X}_\phi$ and $\mathcal{Y}_\psi = \{y \in \mathcal{Y} : \psi\}$ that are restricted by constraints and additionally the reachability set, it can also be defined as

$$\mathcal{R} \subseteq \mathcal{Y}_\psi. \tag{4.41}$$

If Equation 4.41 holds then the input-output properties are satisfied. [Liu+20]

Calculating the reachability set exactly is not scaling well, therefore besides exact reachability there is also a lot of research on approximating the reachability set. This enables the use of reachability methods on bigger and more diverse NNs but often leads to incomplete methods. An example of an exact reachability tool is ExactReach [XTJ17] and examples for approximating reachability tools are Ai2 [Geh+18] and MaxSens [XTJ18]. If over-approximation methods are used the result might not only be that the property is satisfied or not satisfied but can also be *unknown*. Nevertheless, over-approximation methods are still often used as they are much more efficient and the completeness of the algorithm is therefore compromised. Furthermore, they can be implemented in an iterative way that takes a closer look at those *unknown* cases by refining bounds or using an additional complete verifier and can thereby be made complete algorithms.

There are two different approaches to over-approximate: Either the sets can be over-approximated, for example by split-and-join algorithms or interval arithmetic, or the function is over-approximated, e.g. by symbolic propagation. Approximating the sets on reachable outputs of the different layers is done in many tools like NNV [Tra+20] and AVeriNN [BP]. It can, for example, be differed between using naive interval propagation that would for an input x and an output y

$$x \in [0, 1], y = 2x - x \tag{4.42}$$

produce the following bounds

$$y \in [2 * 0 - 1 * 1, 2 * 1 - 1 * 0] = [-1, 2] \tag{4.43}$$

while symbolic interval propagation would recognize that the input x is used twice and that $y = 2x - x = x$, and therefore return

$$y \in [0, 1] \tag{4.44}$$

as bound. Furthermore, it can simplify the verification process a lot if the non-linear activation functions are over-approximated by linear functions. This is also used a lot in optimization-based verifiers and will be described in more detail in subsubsection 4.8.1.3.

Firstly, another look should be taken at the approximation and representation of the reachability sets of each layer which is one of the key challenges in reachability analysis. Often it is assumed that the input space $\mathcal{X}$ is a polytope, zonotope, or some similar structure that is transformed by each of the layers to some new object of the same structure. Research has been done on good-fitting data structures that can be simply transformed by the neuron's operations. The DeepZ [Ryo+21] verification tool uses zonotopes, the NNV [Tra+20] tool uses star sets, and a modified kind of star sets is defined for the AVeriNN tool [BP], the so-called Interval Star set. In addition, tools like NNV [Tra+20] combine reachability tools with different approaches depending on the use case.

The reachability approach can also be included in specialized algorithms like geometric path enumeration as it is done in the verification tool nnenum [Bak21]. Moreover, it can be combined with branch-and-bound methods and optimization techniques that help with finding tighter bounds for the approximation.

4.8.1.2 *SMT-based Solving*

The verification of input-output properties can also be written as a Satisfiability Modulo Theory (SMT) problem. Therefore, it is checked if a formula is satisfiable. In the formula, the function of the neural network and the negation of the property have to be encoded, so the solver checks if there exists an input x with the following properties

$$\exists x \in \mathcal{X} : (x \models \phi) \land (\hat{y} \models \neg\psi) \land (\hat{y} = f(x)) \tag{4.45}$$

based on Equation 4.1. If the solver finds a result, then this result can be taken as a counterexample and proof that the property is not satisfied for all inputs. Early methods for the verification of neural networks with piece-wise linear activation functions used that approach for example Reluplex [Kat+17] and Planet [Ehl17]. Reluplex uses a specialized version of the simplex algorithm to find such an example by adapting the simplex algorithm to special ReLU function constraints between the outputs of the neurons after applying the weights and the outputs of the neurons after applying the activation function. Planet uses a SAT solver and cannot only handle the ReLU function but all piece-wise linear functions as activation functions. Moreover, the methods differ in the way they encode the problem.

4.8.1.3 *Optimization*

The problem defined in Equation 4.1 can also be defined as an optimization problem in a similar way to the SMT-based problem as

$$\min_{x,\hat{y}} o(x, \hat{y}, \mathcal{X}, \mathcal{Y}) \text{ s.t. } x \in \mathcal{X}, \hat{y} \notin \mathcal{Y}, \hat{y} = f(x). \tag{4.46}$$

Some objective function $o(x, y, \mathcal{X}, \mathcal{Y})$ is minimized which is defined differently depending on the used methods. In the constraints of the optimization problem, it was stated that the output $\hat{y}$ does not fulfill the properties that restricted $\mathcal{Y}$. So the goal is to find the case where the property is just not satisfied anymore.

The constraints also have to include the indirect description of the model f to define that the value of $\hat{y}$ is the result of applying f on the input x. The model can be encoded to linear constraints or variations of it like relaxed linear constraints, slack linear constraints, or mixed integer linear constraints. Using methods to convert the model in such constraints and optimizing this complex set of constraints afterward is also referred to as primal optimization. Dual optimization, on the other hand, aims to simplify the constraints beforehand. This transformation to a dual problem is done using relaxation and results in simpler constraints but a more complex objective function. Example tools that are based on primal optimization are NSVerify [HL20] and MIPVerify [TXT19]. For transforming the problem into a dual optimization problem different relaxation methods are used like the Lagrangian Relaxation for the tool Duality [Kri+18] and Semi-definite Relaxation for the tool Certify [RSL20].

The NN function cannot be described by only linear constraints as there is non-linearity in the activation function and therefore, it cannot be written as a Linear Programming (LP). However, by adding integer constraints that can take zero or one as a value, piece-wise linear functions can be represented by Mixed-Integer Linear Programming (MILP). For arbitrary activation functions linear lower and upper bounds can be calculated as done in the tool CROWN [Zha+18]. Besides different mathematical formulas to calculate as tight as possible polytopes around different action functions, the tool α-CROWN [Xu+21] proposed to use a parameter α and a LP to use optimization for getting even tighter bounds. In a later version, the GCP-CROWN [Zha+22b], they also use General Cutting Planes (GCP) for the relaxation of the problem to a dual problem.

Further successful tools that are based on optimization problems are MN-BaB [Fer+22] which implements an efficient dual solver, VeriNet [HL20] which also implements symbolic interval propagation to get a linear approximation, CGDTest [Nag+23], Marabou [Kat+19] - the successor of Reluplex -, and PeregriNN [KFS21].

4.8.1.4 *Search and Branch-and-Bound*

Search algorithms are combined with reachability or optimization in many AI verification tools like ReluVal [Wan+18b], Neurify [Wan+18a], and FastLin [Wen+18]. If used in combination with reachability, they support getting a fast result by searching for counterexamples in the input space or hidden space or by narrowing down the input space that has to be considered. [Liu+20]

Most of the nowadays successful tools use Branch-and-Bound (BaB) approaches to get more efficient like MN-BaB [Fer+22], CROWN [Zha+18] and VeriNet [HL20]. The BaB algorithm consists of two parts, the branching and the bounding. During branching the search space is split into two sub-spaces that are considered separately. For example, branching on ReLU neurons leads to two linear sub-spaces that can be solved more easily. However, as the number of spaces that are obtained by splitting grows exponentially to the number of neurons that one splits on, splitting on all ReLU neurons would lead to scalability issues. Therefore special techniques to decide on which neuron to branch on are developed in tools like VeriNet [HL20]. The branching can also be accelerated using Linear Relaxation-based Perturbation Analysis (LePRA) as done in α-CROWN [Xu+21]. The second part of the BaB algorithm is the bounding which was already discussed in the previous sections.

Research is also done on how the BaB algorithm can be specially adapted to the other approaches and how it can be parallelized on hardware like in the β-CROWN tool.

There are different approaches where optimization problems are combined with search algorithms. Besides searching the input space, the function space can also be searched through by assigning values to the activation function and thereby trying out different activation patterns. One approach would be to do a local and global search to find local and global optima, while another approach would be to do a tree search in the function space.

Moreover, it can also be searched for counterexamples directly like done in the verification tools VerAPAK [DS] and BaB-Attack [Zha+22c]. These tools concentrate on making the search for adversarial examples more effective. One concept here is to search the activation space like done with a top-down beam-search in BaB-Attack [Zha+22c].

4.8.2 *Competition for the Verification of Local Input-Output Properties*

Since 2020 there is a competition called Verification of Neural Networks Competition (VNN-COMP) [LJ; BLJ; Bak+] where verification solutions for Neural Networks are compared. The reports of these yearly competitions [BLJ21; Mü+23; Bri+23] already provide a good overview of the state of the tools for local input-output properties. The most successful tools will be briefly described here.

4.8.2.1 *α-β-CROWN*

The verification tool α-β-CROWN [Zha+] was the winner of the competitions VNN2021 and VNN2022. It is based on optimization and a specialized BaB approach. The tool was composed of many different tools that are described in different research papers.

The first one is the tool CROWN [Zha+18] which implements the idea of bound propagation by providing formulas to calculate piece-wise linear upper and lower bounds for arbitrary activation functions. They extend this to also create piece-wise quadratic bounds. The second one is α-CROWN [Xu+21] which implements a method to get tighter bounds by using a parameter α which represents the slope of the lower bound and is optimized using gradient descent. Furthermore, it introduces the use of Linear Relaxation-based Perturbation Analysis (LiRPA) to accelerate the branching of the BaB algorithm by formulating it as an LP. The third one is β-CROWN [Wan+21] which adds a specialized version of BaB to the CROWN tool. This also enables parallelization and fast execution on GPUs. As a further tool, they developed GCP-CROWN [Zha+22b], a tool that makes use of General Cutting Planes for the relaxation of the LP. Furthermore, it implements the conversion of the problem to a dual problem.

α-β-CROWN is a very powerful tool as it supports FC-NNs, CNNs, RNNs and Transformers, residual connections, average and max pooling, and arbitrary activation functions, for example, ReLU, sigmoid, and hyperbolic tangent (Tanh). The tool successfully took part in all benchmarks of the competition and therefore checked properties on NNs with up to 13.6 Million neurons for different applications like image classification and Reinforcement Learning. However, for all activation functions but ReLU the tool is incomplete.

As verification properties, the tool accepts all linear specifications on the output of the NN. Depending on the benchmark the tool can be used without any additional license, with the Gurobi License [TBB] or the IBM CPLEX license [IBM].

4.8.2.2 *MN-BaB*

Like α-β-CROWN the tool MN-BaB [Fer+22] is also based on optimization and the BaB algorithm. It converts the problem into a dual problem and efficiently solves it with a dual solver that runs on a GPU. Its main contribution is a Multi-Neuron Guided Branch-and-Bound method that enables the calculation of tighter multi-neuron constraints for guidance on branching.

The tool got the second prize in last year's competition VNN-COMP 2022. It supports fully connected NNs, CNNs, and residual networks and different activation functions like ReLU, Sigmoid, Tanh, and Max pooling. However, it also needs a Gurobi license [TBB].

4.8.2.3 *VeriNet*

VeriNet [HL20] is another successful verification tool that is based on Symbolic Interval Propagation (SIP). Like the previous tools, it also implements a special kind of BaB with efficient branching.

VeriNet can also handle FC-NNs, CNNs, and residual Networks. It works with different activation functions including ReLU, Sigmoid, and Tanh. The tool needs an Xpress Solver [Fai] license when used on large NNs.

4.8.2.4 *Marabou*

The tool Marabou also took part in the VNN-COMP. In only got the 7th place in the competition in 2022, however, it still should be introduced as it runs without an Nvidia GPU and therefore will play a role in the use case demonstration of the framework in chapter 5. Marabou is the successor of the early tool Reluplex which extended the simplex linear optimization algorithm by ReLU constraints. Like that, Marabou is also SMT-based but it additionally uses a divide-and-conquer algorithm for more efficiency and supports other piece-wise linear activation functions besides ReLU and more input formats for neural networks and properties as well.

4.8.3 *Approaches for the Verification of Global Input-Output Properties*

By global input-output properties, global robustness, invariance properties, individual fairness, global safety, and correctness properties are included. There is much less research on them than on local input-output properties and most research focuses directly on global stability or robustness. The global input-output properties differ from the local ones by setting constraints on more than one input and the corresponding output. For the properties regarded here, the constraints are about the relations of two inputs x and x' and their outputs $f(x)$ and $f(x')$. However, for this general definition of global properties there do not exist general approaches. Therefore, approaches for the specific sub-properties are discussed in this section.

4.8.3.1 *SAT-based Fairness*

The tool Fairify [BR22] enables the verification of individual fairness of NNs with structured input, i.e. inputs with clearly identifiable discriminating input neurons. The approach they use is to encode the preconditions and the inverse of the post-condition as SAT formulas and solving it with the SMT solver Z3 [BNW]. This is similar to the SMT-based approaches for verifying local input-output properties. However, two different inputs x and x' have to be considered here and both of them and their NN processing steps have to be modeled. As this makes the problem much more complex, the tool reduces the complexity of the NNs and the input space beforehand. In order to do that, it makes use of the concepts of partitioning the input space and pruning the NNs for the different partitions independently. To reduce the network complexity by leaving out neurons that are zero for all or most inputs they combine two approaches: Sound pruning and heuristic-based pruning. They define a timeout of 100 seconds for the sound pruning and SAT solving of each partition and if the result is *unknown* afterwards, the heuristic approach is executed.

4.8.3.2 *MILP-based Global Stability*

When the tool Reluplex [Kat+17] was proposed, they also mentioned that it is extensible to global stability by encoding two copies on the NN, N_1 and N_2 which operate on the two inputs, x_1 and its perpetuated version x_2, respectively. The thereby produced problem of the form of Equation 4.10 is then put in the Simplex-based solver that was extended by ReLU nodes. However, as twice as many neurons with ReLU activation have to be considered compared to local stability, this approach does not scale.

Cheng, Nührenberg, and Ruess [CNR17] propose to define global robustness for classification problems by involving the probabilities for the classes in the output. For all points in the input space x that strongly classify to a class m according to a parameter, the outputs for similar inputs x' with a perturbation smaller than δ_m should include the at least k highest probability for the class m. They model the problem of determining the perturbation bounds δ_m as a MILP problem.

By Wang, Huang, and Zhu [WHZ22] a special encoding for the two copies of a NN fed with different inputs is proposed called interleaving twin-network encoding. Therefore, there is not one encoding of the NN for each of the two inputs, but the NN is encoded only one time. However, this encoding is extended for each connection between two neurons by a connection of their different values for the inputs x and x', done with simple addition and subtraction operators. Furthermore, they adapt the over-approximation techniques that were proposed by Huang, Fan, Chen, et al. [Hua+20a] for local stability verification to be also used for global verification. This technique is based on network decomposition which splits the NN into smaller parts to handle those less complex parts more easily and LP relaxation which aims to approximate the nonlinear ReLU function by a linear function. Their special network encoding and approximation techniques lead to much better efficiency and scalability than previous approaches.

4.8.3.3 *Search-based Fairness*

Urban, Christakis, Wüstholz, and Zhang [Urb+20] propose a tool called Libra for fairness verification of classification NNs with ReLU as an activation function, composed of two parts: a forward and a backward pass. Their approach relies on the idea that the verification process can be parallelized by splitting the input space based on activation patterns and solving the fairness problem on each of them individually. The forward pass groups the activation pattern into so-called abstract activation patterns and then identifies the input region belonging to it. Based on that the input space is split into regions. Afterwards, the backward pass tries to solve the fairness problem for each abstract activation pattern in parallel. This is done by looking at the output of a specific class and leading it back to its corresponding inputs, one time with respect to a sensitive attribute and one time without. The distinction of the sub-input spaces are then taken, to determine if the NN is fair.

4.8.3.4 *Lipschitz Constant Estimation*

Some research in the area of formal verification of global properties on ML focuses specifically on global stability and robustness. As described in subsubsection 4.5.3.3, this can also be defined by Lipschitz constants.

Many different formulas for calculating an as tight as possible Lipschitz constant θ have been proposed in the literature. A simple way to determine a Lipschitz constant for a feed-forward network with m layers with weights W_i and a non-expansive activation function is by multiplying the spectral norms of the weight matrices:

$$\theta_{simple} = \Pi_{j=1}^{m} \|W_j\|. \tag{4.47}$$

However, the thereby obtained bound is often loose and useless.

Khromov and Singh [KS23] propose to calculate the Lipschitz constant by

$$\theta_{Combettes} = \sup_{\Lambda_1 \in \mathcal{D}_{N_1}, \ldots, \Lambda_{m-1} \in \mathcal{D}_{N_{m-1}}} \|W_m \cdot \Lambda_{m-1} \cdot \ldots \cdot \Lambda_1 \cdot W_1\|. \tag{4.48}$$

where $\mathcal{D}_{N_j}$ is the set of diagonal matrices of size N_j with zeros or ones as values. They show that $\theta_{Combettes} \leqslant \theta_{simple}$ and therefore that their formula often leads to smaller and better Lipschitz constants.

Another approach was proposed by Zhang, Jiang, He, and Wang [Zha+22a] and is based on solving a semidefinite program. Furthermore, in the constraints of the optimization problem, they encode an over-approximation of the non-linear activation function in the form of so-called incremental quadratic constraints.

The scalability of the previous approach was increased by Xue, Lindemann, Robey, et al. [Xue+22] by splitting the Linear Matrix Inequality (LMI) into smaller pieces because these pieces can be solved much faster.

Pauli, Gramlich, and Allgöwer [PGA23] propose an approach to adapt this method to one-dimensional CNNs while making use of its structure. In addition, they split the LMI into smaller LMIs for each layer.

4.8.3.5 *Further Approaches*

The tool DeepSafe [Gop+18a] is based on an approach that first uses the clustering algorithm k-means to divide the input space into regions based on the labels the different inputs in the training data had. Each region has a center and a radius and they can be fed into a local verification algorithm to check if the region is robust. This advantage of being able to use a more efficient local verification tool comes with the disadvantage that only within one region global robustness can be proven and for classification problems with many classes, it is also not scaling well.

The tool DeepTRE [Rua+19] determines lower and upper bounds on the safe radius which define the maximum perturbation that can be added to still get the right label and how much perturbation will already lead to an adversarial example. These bounds are iteratively improved. They measure the difference through the perturbation by the Hamming distance, also called the L_0 norm, which is especially suitable for perturbations in the form of a few changed pixels.

There also exist methods for directly improving and not certifying properties like fairness. An extensive summary of such improvement methods for fairness and insights on when to apply which method is given in [ZS22] and [Che+23].

Pauli, Koch, Berberich, et al. [Pau+22a] propose a method, that integrates the aim of getting small Lipschitz bound directly into the training. They specify an optimization scheme that does not only take the loss function into account but also the robustness of the NN defined by a Lipschitz constant that is as small as possible.

4.8.4 *Approaches for the Verification of Probabilistic Properties*

Even if probabilistic properties are often not directly counted to the formal properties, they can give formal guarantees on probabilities. However, the approaches are often still requirement specific nowadays.

4.8.4.1 *Probabilistic Robustness*

Probabilistic robustness was defined by Mangal, Nori, and Orso [MNO19] as requiring the probability for f being globally robust to be higher than some threshold. In the same paper, they also provide a solution on how probabilistic robustness can be determined by using abstract interpretation and importance sampling. By abstract interpretation, it is meant that the behavior of the neural network is approximated. Importance sampling is a special statistical sampling technique that enables the handling of unlikely events based on fewer samples than needed in naive sampling by focusing on the space of unlikely events.

A further tool for the verification of probabilistic robustness is CC-Cert [Pau+22b] which is based on the Chernoff-Cramer bounds. They concentrate on perturbations resulting from practical transformations like rotation and translation of images.

Weng, Chen, Nguyen, et al. [Wen+19] propose a tool called PROVEN which works on top of a local input-output property verification tool and additionally provides probabilistic robustness certification with relatively small overhead. This is of interest for those cases where not only the worst-case bounds are relevant but also how likely robustness is for a given distance. They provide formulas for the calculation of lower and upper bounds of this probability.

4.8.4.2 *Generalization*

The generalization bound can directly be calculated by one of its mathematical definitions. To test the generalization property a maximum generalization gap δ_{max} and ϵ have to be set. Then δ can be calculated and if it is smaller than δ_{max} the generalization property is verified.

4.8.4.3 *Probabilistic Fairness*

Albarghouthi, D'Antoni, Drews, and Nori [Alb+17] propose a tool called FairSquare which can handle the verification of probabilistic properties, and specializes in probabilistic fairness properties on different kinds of decision-making programs. Their verification method is based on so-called weighted volumes which is a way of computing an integral over boolean formulas.

However, as this approach is really slow, Bastani, Zhang, and Solar-Lezama [BZSL19] proposes a more efficient tool for verifying group fairness properties like demographic parity, equal opportunity, and path-specific causal fairness. The tool VeriFair approximates the expected output value of the minority and the majority group and provides probabilistic guarantees by using adaptive concentration inequalities.

Several further platforms that implement different fairness metrics, benchmark datasets, and algorithms for fairness measurement have been developed in the last years. They often do not only address post-processing verification but also pre- and in-processing verification. However, for the verification of the trained model they often only provide verification measurement algorithms for group fairness properties, like the frameworks Fairlean [Aga+18], AIF360 [Bel+18], and Aequitas [Sal+19], or are only statistical testing based like FairTest [Tra+16] and AuditAI [Dia+].

4.8.5 *Approaches for the Verification of Relations between different Models*

When considering equivalence properties, the outputs of the two different models have to be compared in the constraint. Different approaches have been developed to handle the thereby much more complex problem.

Paulsen, Wang, and Wang [PWW20] propose a tool called ReluDiff for finding differences between two NNs of a similar structure. It consists of a forward pass that makes use of the structural similarity of the NNs and determines bounds on the difference and a backward pass that refines the difference.

Another approach is to encode the problem as a MILP problem. This is done in the tool MILPEQUIV [KBKS20]. However, encoding two networks often leads to scalability errors.

Teuber, Büning, Kern, and Sinz [Teu+21] extend the idea of using Geometric Path Enumeration to take two NNs as input. This reachability approach was originally proposed for local input-output property verification and was already mentioned in subsubsection 4.8.1.1. In their extension, they propagate the star sets through the first NN and then they restrict the input set based on the obtained output star set before using it as the input for the other NN.

4.9 THE METHOD SELECTION PROCESS

As there are many methods to choose from, support is needed especially in this step. All fitting methods regarding the attributes of the object and the property to verify are returned by the function. An opaque behavior in Jython is defined for that purpose in the verification package in Cameo. It creates *VTask* objects for all *toProofOn* connections, extracts the necessary information on the object and property, and uses them as input for the execution of an external Python script.

This script can also be executed independently of the whole Cameo-based framework to find fitting methods. Moreover, having an external Python script allows the use of Python libraries like Pandas [The20] in contrast to the Jython language that is included in Cameo's opaque behaviors. The Python script basically reads the CSV file *tools.csv*, which includes the information on the tools, into a Pandas dataframe and then filters this dataframe by certain criteria. The whole CSV file can be found in Appendix A.

Every row includes information on one tool. The columns describe different characteristics of the tool like which types of objects and which properties to verify it supports. The names of the columns consist of three parts. The first part defines if the characteristic restricts the object of verification (obj), the property of verification (prop), or is a constraint on the verification (treq) process. The second part is one of the three operators *need* (n), *support* (s), and *compare* (c) and the third part describes the characteristic. Examples would be *obj:s:activations*, *obj:c:no_of_neurons*, and *treq:n:licenses*. In the determineMethod() Python script the corresponding information on the object, the property, and the verification constraints of the modeled system is compared to these characteristics of each tool. For example, if the model has ReLU and Tanh as activation functions, for each tool the script checks if both are in the list of supported activation functions. So the activations of the system should be a subset of those supported by the tool. The need-operator works the other way around. If the verification hardware has only a Central Processing Unit (CPU) but the tool also needs a Nvidia GPU it cannot be used. Therefore, those characteristics are checked for if the tool's properties are a subset of what the system provides. The third operator is the compare operator. If a tool is known to support FC-NNs with at least up to 13 million neurons then the number of neurons in the system has to be less or equal to use the tool. These operators allow to add further characteristics and automatically evaluate them in the script.

It should be mentioned that the table only captures what was already tested with this tool yet. Therefore, the number of neurons that is listed as maximum is not a hard limit but instead just the largest number for which a test was executed in the literature. In addition, in section 6.2 further aspects will be discussed on how meaningful measures like the number of neurons are without looking at the detailed structure. Moreover, the handling of umbrella terms will be discussed in that section, too. For example, tools might be able to handle all piece-wise linear activation functions. If the defined NN uses only ReLU activation functions, the information that ReLU is piece-wise linear has to be mapped on the problem.

Expressing more complex things about the abilities of tools, like that a tool needs only a CPU for ReLU-based NNs and for other activation functions also needs a NvidiaGPU, can at the moment only be done by writing two rows for the tool, one with only ReLU as activation and CPU only as hardware and the other row concluding all activation functions and CPU and GPU.

All methods that were determined as fitting to the problem are saved in the attribute *matchingMethods* of the *VerificationTask* object. The first method is proposed in the *choosenMethod* attribute but this can be changed by the user to one of the other methods.

4.10 VERIFICATION TOOL EXECUTION

The automatic execution of all the verification tools that were selected is out of the scope of this thesis. However, one tool, Marabou, was installed to show the workflow. However, the Laptop on which the tool was executed only has Window installed. Therefore, the version of Marabou from 2019 was used because it is the latest one that support installation on Windows. This version will be called Marabou-Win in the rest of the thesis.

When the *RunVerificationTools()* method is executed, those verification tasks, that have Marabou-Win as the chosen method and filenames defined containing the NNs definition and the formal definition of the property, will be verified by the tool.

The script iterates through all verification tasks and executes the verification tool if possible. Then, it reads the output of the execution after the tool has finished or after a timeout of 150 seconds. The whole console output is saved in the *result_comment* attribute of the verification task and if the result is *sat* or *unsat*, the *result* attribute is set to true or false respectively. In case of a timeout, the result is set to *None*.

4.11 TRACING BACK THE RESULTS ON REQUIREMENTS

As for systems with many objects to check and many properties to verify the list of verification tasks can become really long and confusing, their results should be mapped back to the requirements to which their property to verify belongs to. If all *VerificationTasks* that are thereby connected to a requirement have true as a result, the requirement is verified, if at least one is false, the requirement cannot be verified. Unknown results can appear if there is no false result for Vtasks but some unknowns. The opaque behavior *MapResultsVtasksToReqs()* can be started for automatically getting this mapping.

5 APPLYING THE FRAMEWORK ON USE CASES IN AVIONICS

This chapter will show how the framework can be applied based on two example use cases from avionics: The famous benchmark of the HCAS of ACAS Xu [Jul+16] and a threat localization systems for airplanes [Spr+23].

The framework is applied to the use cases according to the flow presented in Figure 4.8. The template includes the package structure presented in Figure 4.1 with the initialized diagrams.

5.1 HORIZONTAL COLLISION AVOIDANCE SYSTEM

Aircraft collision avoidance systems were developed in the ACAS X family based on a Markov Decision Process (MDP) and a given decision logic. However, these systems need a lot of storage space for the MDP table. Therefore, Julian, Lopez, Brush, et al. [Jul+16] proposed to compress the table of the horizontal version ACAS Xu by NNs. These NNs were often used as benchmarks for simple safety-critical systems like in [Kat+17], [Kat+19], and as unscored benchmark in the VNN-COMP [Mü+23].

5.1.1 *Introduction to HCAS*

For understanding the inputs and outputs of the NN, one has to take a look at the original ACAS Xu logic table. The following parameters are considered:

- ▷ ρ: distance between ownship and intruder (m)
- ▷ v_{own}: speed of ownship (m/s)
- ▷ v_{int}: speed of intruder (m/s)
- ▷ θ: bearing angle to intruder (rad)
- ▷ ψ: heading angle of intruder relative to ownship (rad)
- ▷ $a_{prev} \in [COC, WL, WR, SL, SR]$: previous advisory (Clear Of Conflict (COC), Weak Left (WL), Weak Right (WR), Strong Left (SL), Strong Right (SR))
- ▷ τ: time until the airplanes are on the same vertical level (s)

For each of the parameter a set of possible values is defined, for example τ can be in $[0, 1, 5, 10, 20, 40, 60, 80, 100]$. However, since there are many parameters that can take many different values and all combinations of those values are listed, the table is very big. Based on the values of these parameters of every entry, the best next advisory is given in the table.

The logic table is split by the values of τ and a_{prev} and for each of their values a NN is trained that learns to output the right advisory action dependent on the values of the other parameters. The structure of the NNs and the whole system will be regarded closer in the next section.

5.1.2 *Modeling the System Structure*

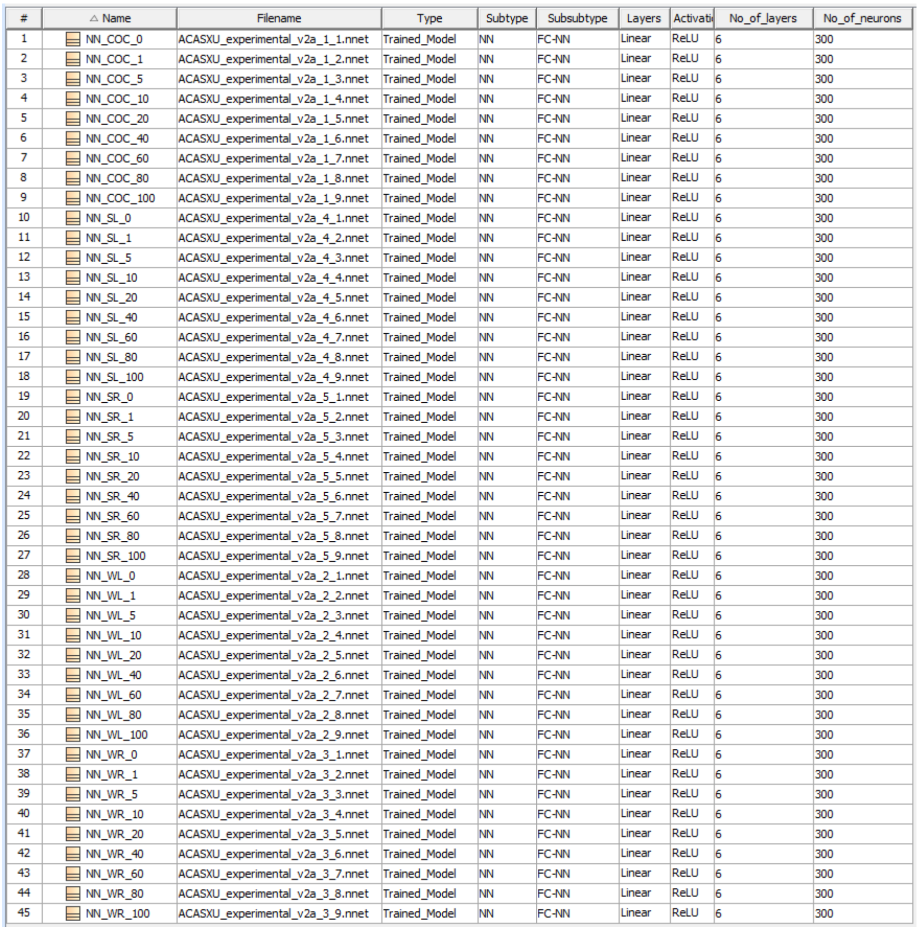
The HCAS system can be modeled by a learning component and an end-system component. The learning includes the pre-processing of the data, for example, by splitting the table based on τ and a_{prev}, and one for the initialization and training of the NNs. The table containing all NNs is shown in Figure 5.1. They are named by the previous advisory and the time until the vertical distance is zero. They are trained models, more specific FC-NNs and only have linear layers with ReLU as activation function. Each NN has 300 neurons, arranged on 6 layers. [Jul+16]

The NNs can be defined as blocks of type *Neural_Network* in the system structure BDD. However, since there were so many NNs in the system, the NNs and their relations were created by an opaque behavior for this use case and the relations were then pulled into the system structure diagram. The nnet files that contain the NNs where downloaded from the work of Katz, Huang, Ibeling, et al. [Kat+]. For every NN the filename attribute has to be defined in the NN objects for the automatic execution of the verification later.

5.1.3 *Defining the System Requirements*

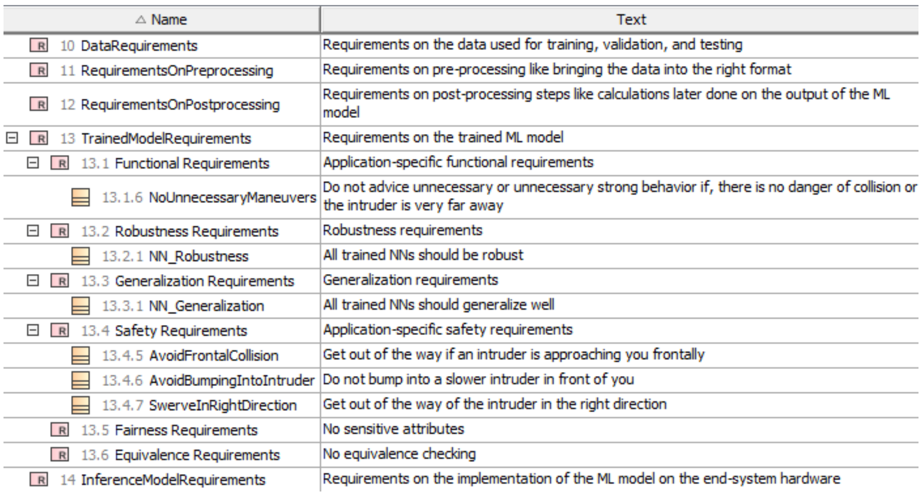
The basic kinds of requirements were already defined in the framework and concerning the trained model nested system requirements were defined.

As correctness requirement, it was specified that the system should not do extensive unnecessary maneuvers. For example, there should not be a SL, advice then SR, and then SL again. Furthermore, the robustness property should hold for all NNs. The same goes for generalization. Moreover, different safety requirements have to be defined like avoiding frontal collisions, avoiding to tailgate, and moving in the wrong direction when trying to get out of the way of the intruder.



#	△ Name	Filename	Type	Subtype	Subsubtype	Layers	Activati	No_of_layers	No_of_neurons
1	NN_COC_0	ACASXU_experimental_v2a_1_1.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
2	NN_COC_1	ACASXU_experimental_v2a_1_2.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
3	NN_COC_5	ACASXU_experimental_v2a_1_3.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
4	NN_COC_10	ACASXU_experimental_v2a_1_4.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
5	NN_COC_20	ACASXU_experimental_v2a_1_5.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
6	NN_COC_40	ACASXU_experimental_v2a_1_6.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
7	NN_COC_60	ACASXU_experimental_v2a_1_7.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
8	NN_COC_80	ACASXU_experimental_v2a_1_8.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
9	NN_COC_100	ACASXU_experimental_v2a_1_9.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
10	NN_SL_0	ACASXU_experimental_v2a_4_1.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
11	NN_SL_1	ACASXU_experimental_v2a_4_2.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
12	NN_SL_5	ACASXU_experimental_v2a_4_3.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
13	NN_SL_10	ACASXU_experimental_v2a_4_4.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
14	NN_SL_20	ACASXU_experimental_v2a_4_5.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
15	NN_SL_40	ACASXU_experimental_v2a_4_6.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
16	NN_SL_60	ACASXU_experimental_v2a_4_7.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
17	NN_SL_80	ACASXU_experimental_v2a_4_8.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
18	NN_SL_100	ACASXU_experimental_v2a_4_9.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
19	NN_SR_0	ACASXU_experimental_v2a_5_1.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
20	NN_SR_1	ACASXU_experimental_v2a_5_2.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
21	NN_SR_5	ACASXU_experimental_v2a_5_3.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
22	NN_SR_10	ACASXU_experimental_v2a_5_4.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
23	NN_SR_20	ACASXU_experimental_v2a_5_5.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
24	NN_SR_40	ACASXU_experimental_v2a_5_6.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
25	NN_SR_60	ACASXU_experimental_v2a_5_7.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
26	NN_SR_80	ACASXU_experimental_v2a_5_8.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
27	NN_SR_100	ACASXU_experimental_v2a_5_9.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
28	NN_WL_0	ACASXU_experimental_v2a_2_1.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
29	NN_WL_1	ACASXU_experimental_v2a_2_2.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
30	NN_WL_5	ACASXU_experimental_v2a_2_3.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
31	NN_WL_10	ACASXU_experimental_v2a_2_4.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
32	NN_WL_20	ACASXU_experimental_v2a_2_5.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
33	NN_WL_40	ACASXU_experimental_v2a_2_6.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
34	NN_WL_60	ACASXU_experimental_v2a_2_7.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
35	NN_WL_80	ACASXU_experimental_v2a_2_8.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
36	NN_WL_100	ACASXU_experimental_v2a_2_9.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
37	NN_WR_0	ACASXU_experimental_v2a_3_1.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
38	NN_WR_1	ACASXU_experimental_v2a_3_2.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
39	NN_WR_5	ACASXU_experimental_v2a_3_3.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
40	NN_WR_10	ACASXU_experimental_v2a_3_4.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
41	NN_WR_20	ACASXU_experimental_v2a_3_5.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
42	NN_WR_40	ACASXU_experimental_v2a_3_6.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
43	NN_WR_60	ACASXU_experimental_v2a_3_7.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
44	NN_WR_80	ACASXU_experimental_v2a_3_8.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300
45	NN_WR_100	ACASXU_experimental_v2a_3_9.nnet	Trained_Model	NN	FC-NN	Linear	ReLU	6	300

Figure 5.1: List of all Neural Networks of HCAS

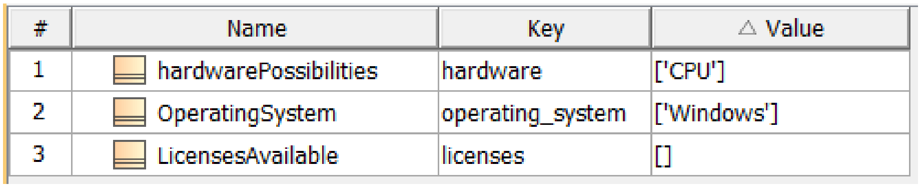


△ Name	Text
10 DataRequirements	Requirements on the data used for training, validation, and testing
11 RequirementsOnPreprocessing	Requirements on pre-processing like bringing the data into the right format
12 RequirementsOnPostprocessing	Requirements on post-processing steps like calculations later done on the output of the ML model
13 TrainedModelRequirements	Requirements on the trained ML model
13.1 Functional Requirements	Application-specific functional requirements
13.1.6 NoUnnecessaryManeuvers	Do not advice unnecessary or unnecessary strong behavior if, there is no danger of collision or the intruder is very far away
13.2 Robustness Requirements	Robustness requirements
13.2.1 NN_Robustness	All trained NNs should be robust
13.3 Generalization Requirements	Generalization requirements
13.3.1 NN_Generalization	All trained NNs should generalize well
13.4 Safety Requirements	Application-specific safety requirements
13.4.5 AvoidFrontalCollision	Get out of the way if an intruder is approaching you frontally
13.4.6 AvoidBumpingIntoIntruder	Do not bump into a slower intruder in front of you
13.4.7 SwerveInRightDirection	Get out of the way of the intruder in the right direction
13.5 Fairness Requirements	No sensitive attributes
13.6 Equivalence Requirements	No equivalence checking
14 InferenceModelRequirements	Requirements on the implementation of the ML model on the end-system hardware

Figure 5.2: System Requirements Table of HCAS

5.1.4 *Defining the Technical Requirements*

The laptop in use only has a CPU and an integrated GPU. Because GPUs were only listed if they are Nvidia GPUs, just the CPU is listed here. In addition, there is no access to solver licenses. Furthermore, only Windows ins installed on the laptop. The table of the technical requirements is shown in Figure 5.3.



#	Name	Key	△ Value
1	hardwarePossibilities	hardware	['CPU']
2	OperatingSystem	operating_system	['Windows']
3	LicensesAvailable	licenses	[]

Figure 5.3: Verification Requirements Table of HCAS

5.1.5 *Deriving the Properties to Verify*

Katz, Barrett, Dill, et al. [Kat+17] already defined ten properties, ϕ_1 to ϕ_{10}, that should hold on the HCAS system. These properties are local input-output properties. The output of the NNs consists of five values, one for each possible action, and the lower the value the better the action. Because they fit to the correctness and safety requirements defined above, they are also used here.

Several properties can be marked as refinements of the correctness requirement *NoUnnecessaryManeuvers*. To this, the first property ϕ_1 fits because it states that the output value of COC should be below a threshold if the intruder is far away and the speed of the ownship is much lower than the one of the intruder. The property ϕ_2 deals with the same situation requiring the output value for COC to be not the maximal of all output values. The properties ϕ_6 and ϕ_{10} add that for very big distances between ownship and intruder and specific previous advisories and values of τ, the best action should be COC. In addition, no strong or rapidly direction changing advisories should be given if the intruder has a big enough vertical distance, as stated by ϕ_7 for the NN with $\tau = 100$ and $a_{prev} = COC$ and by ϕ_8 for the NN with $\tau = 100$ and $a_{prev} = WL$.

The safety requirement *AvoidFrontalCollision* can be refined by properties ϕ_3 and ϕ_9. Property ϕ_3 states that if the intruder is directly ahead and moving towards the ownship, then COC should not be the best action. Property ϕ_9, on the other hand, deals with the situation where the intruder is slightly right in front of the ownship a moves in the opposite direction than the ownship. The property states, that even if the last advisory was weak right, the next best action should be strong left.

Another safety requirement is, that one should not bump into an intruder that is flying in front of the ownship. This is formulated in ϕ_4

for all NNs except those that had COC as last advisory and have at least 60 seconds left until loss of vertical separation.

An example property for the safety requirement of swerving to the right direction is ϕ_5. It deals with the situation that the intruder is coming from the left and is already quite close. Then SR should be given as advisory.

Further requirements that have not been refined yet, are the robustness and the generalization requirements. For the kind of robustness, Table 4.1 is used as guideline. Since the goal of the NN is to represent the logic table, it is enough to take a look at inputs from the ODD. This leads to the choice of stability instead of general robustness. Around the points which are given in the ACAS Xu table, the NN should be robust. So local robustness is regarded. Checking if the network is always robust is necessary here because probabilistic guarantees on that it is robust for most inputs are not enough for such a safety-critical system. This is especially the case for safety properties. For correctness properties on the other hand one can argue that probabilistic guarantees might be enough. Depending on what post-processing steps for deciding on what action to do will follow, distinct robustness might also be a good choice. However, it is assumed for now that simply the best action based on the network's output is executed. Therefore, local non-probabilistic non-distinct stability is chosen as property to verify. This leads to the possibility of expressing the robustness property by a local input-output property. It will be checked for L_∞-perturbations here, and therefore that norm will be used. The property should be checked on all NNs.

For checking the generalization abilities the estimation of generalization bounds should be used. A probability of $(1 - \alpha)$ is defined for which the calculated bound should hold and compare this bound to a maximum generalization gap $g_m ax$. The generalization property should also be checked on all NNs.

Since the HCAS system does not deal with sensitive attributes and no two ML models have to be compared, no fairness and equivalence properties have to be defined.

Figure 5.4 shows how the properties refine the requirements in the *RequirementPropertyRelation* diagram. In this diagram the properties and the refine relations were created. Once the properties are defined in this diagram, they can also be seen in the *PropertiesToVerify* table. Either in this table or by opening their specification in the previous diagram, their attributes can be defined. Depending on what attributes are visualized in it, this table can get very big, still, an example entry is shown in Figure B.1. The editing of a properties specification can be seen in Figure 5.5.

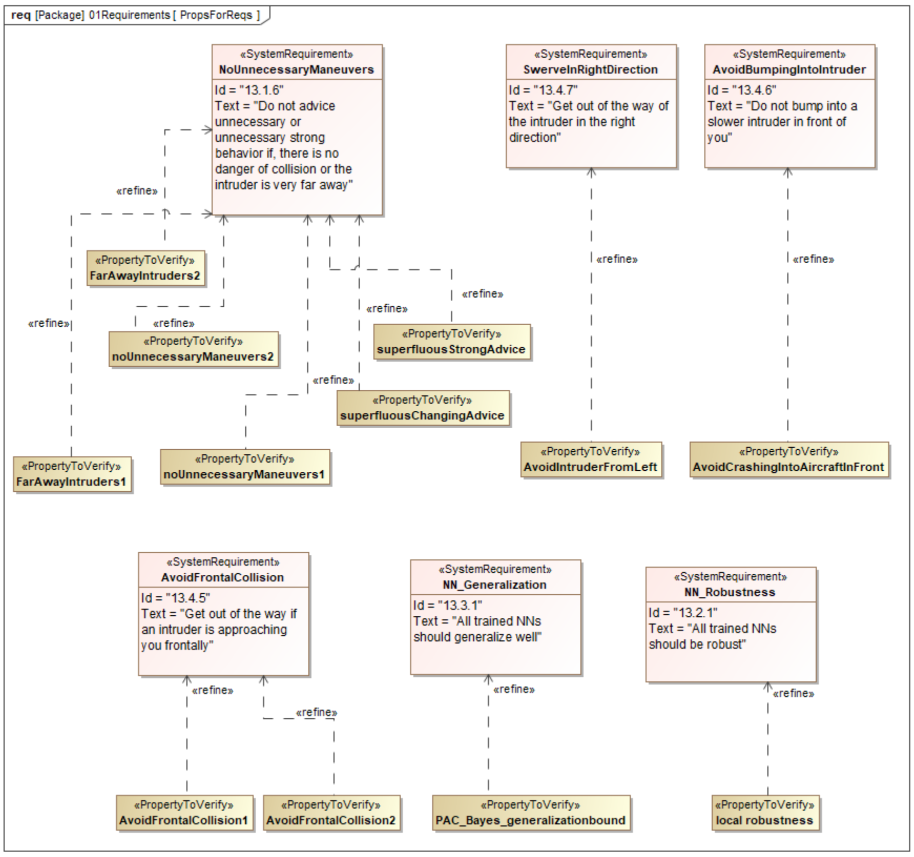
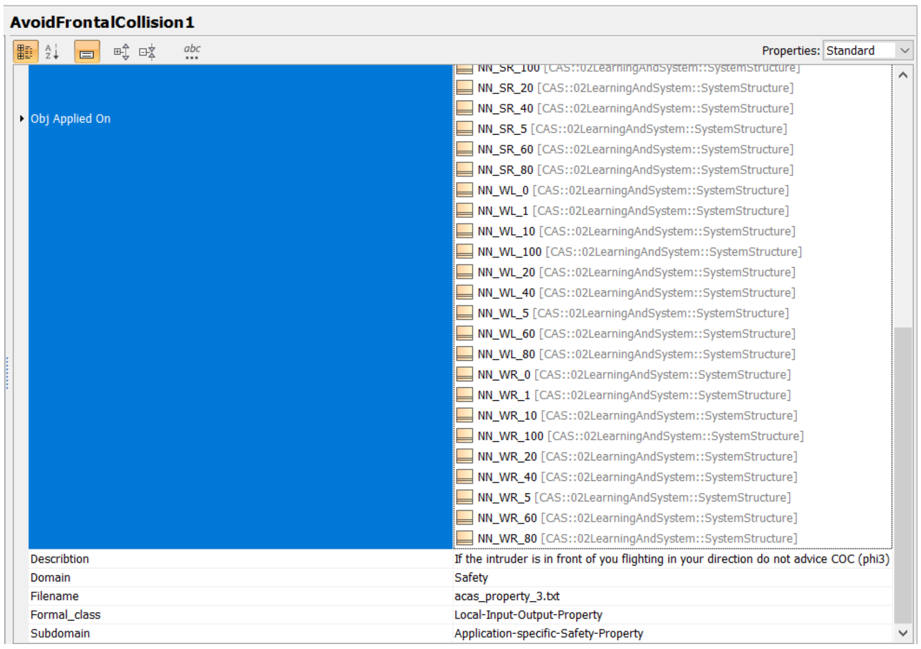

req [Package] 01Requirements [PropsForReqs]
«SystemRequirement» NoUnnecessaryManeuvers
Id = "13.1.6"
Text = "Do not advice unnecessary or unnecessary strong behavior if, there is no danger of collision or the intruder is very far away"
«SystemRequirement» SwerveInRightDirection
Id = "13.4.7"
Text = "Get out of the way of the intruder in the right direction"
«SystemRequirement» AvoidBumpingIntoIntruder
Id = "13.4.6"
Text = "Do not bump into a slower intruder in front of you"
«refine»
«PropertyToVerify» FarAwayIntruders2
«PropertyToVerify» noUnnecessaryManeuvers2
«PropertyToVerify» superfluousStrongAdvice
«PropertyToVerify» superfluousChangingAdvice
«PropertyToVerify» FarAwayIntruders1
«PropertyToVerify» noUnnecessaryManeuvers1
«PropertyToVerify» AvoidIntruderFromLeft
«PropertyToVerify» AvoidCrashingIntoAircraftInFront
«SystemRequirement» AvoidFrontalCollision
Id = "13.4.5"
Text = "Get out of the way if an intruder is approaching you frontally"
«SystemRequirement» NN_Generalization
Id = "13.3.1"
Text = "All trained NNs should generalize well"
«SystemRequirement» NN_Robustness
Id = "13.2.1"
Text = "All trained NNs should be robust"
«PropertyToVerify» AvoidFrontalCollision1
«PropertyToVerify» AvoidFrontalCollision2
«PropertyToVerify» PAC_Bayes_generalizationbound
«PropertyToVerify» local robustness

Figure 5.4: Requirements Refinement of HCAS

After the properties with their formal classes, domains, subdomain, and list of objects they are applied on are defined. They have to be connected to these objects by a *toProofOn* dependency which is done by executing the *GenerateRelationsPropsToObjs* opaque behavior.

5.1.6 *Determining the Methods and Verification Tasks*

Next, the opaque behavior *DetermineMethods* is executed and the verification tasks that it creates were listed in the *VerificationTask* table. The complete table can be found in section B.1. In the column *matchingMethods* all the tools are listed that the script determined as matching to the verification problem. In the column *chosenMethod* the first element of the list is suggested. However, this can be changed in the next step to some other method if another method should be used. Marabou is left here as choice for the local input-output verification. For more information on why other methods were not selected as fitting to the problem, the column *DetermineMethodsConsoleOutput* can be shown in the table. Examples of these outputs for a local input



AvoidFrontalCollision1

Properties: Standard

Obj Applied On

NN_SR_100 [CAS::02LearningAndSystem::SystemStructure]
NN_SR_20 [CAS::02LearningAndSystem::SystemStructure]
NN_SR_40 [CAS::02LearningAndSystem::SystemStructure]
NN_SR_5 [CAS::02LearningAndSystem::SystemStructure]
NN_SR_60 [CAS::02LearningAndSystem::SystemStructure]
NN_SR_80 [CAS::02LearningAndSystem::SystemStructure]
NN_WL_0 [CAS::02LearningAndSystem::SystemStructure]
NN_WL_1 [CAS::02LearningAndSystem::SystemStructure]
NN_WL_10 [CAS::02LearningAndSystem::SystemStructure]
NN_WL_100 [CAS::02LearningAndSystem::SystemStructure]
NN_WL_20 [CAS::02LearningAndSystem::SystemStructure]
NN_WL_40 [CAS::02LearningAndSystem::SystemStructure]
NN_WL_5 [CAS::02LearningAndSystem::SystemStructure]
NN_WL_60 [CAS::02LearningAndSystem::SystemStructure]
NN_WL_80 [CAS::02LearningAndSystem::SystemStructure]
NN_WR_0 [CAS::02LearningAndSystem::SystemStructure]
NN_WR_1 [CAS::02LearningAndSystem::SystemStructure]
NN_WR_10 [CAS::02LearningAndSystem::SystemStructure]
NN_WR_100 [CAS::02LearningAndSystem::SystemStructure]
NN_WR_20 [CAS::02LearningAndSystem::SystemStructure]
NN_WR_40 [CAS::02LearningAndSystem::SystemStructure]
NN_WR_5 [CAS::02LearningAndSystem::SystemStructure]
NN_WR_60 [CAS::02LearningAndSystem::SystemStructure]
NN_WR_80 [CAS::02LearningAndSystem::SystemStructure]

Describtion	If the intruder is in front of you flighting in your direction do not advice COC (phi3)
Domain	Safety
Filename	acas_property_3.txt
Formal_class	Local-Input-Output-Property
Subdomain	Application-specific-Safety-Property

Figure 5.5: Example Property Specification of HCAS

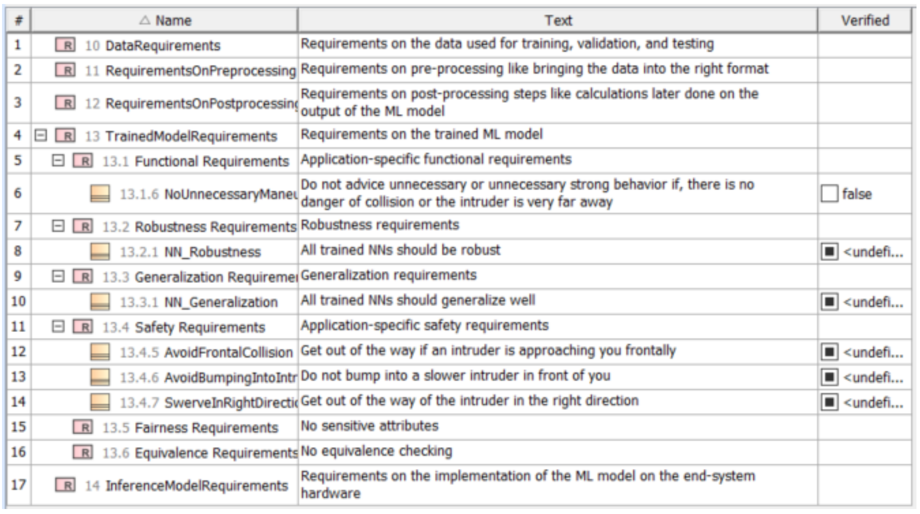
output property and a generalization bound are given in section B.1. The *DetermineMethod()* function outputs the reasons why each of the tools dropped off from the list of matching tools.

Next, one can check in the tools table what file formats Marabou-Win supports as input. Marabou-Win takes NNs in the nnet file format and properties in a specially structured txt-file. To demonstrate the verification flow, for properties ϕ_1 to ϕ_4 the files containing the mathematical formulation were downloaded from Katz, Huang, Ibeling, et al. [Kat+] and the filenames were added to the model as attribute of the properties. For all verification tasks where a tool was chosen, a filename for the property, and a filename for the object were defined, the verification is executed when running the opaque behavior *RunVerificationTasks*. Property ϕ_1 was applied to all NNs, property ϕ_2 to all besides those that had COC as last previous advisory, and property ϕ_3 and ϕ_4 to all besides those that had COC as last advisory and have more than 60 seconds left until loss of vertical separation. So overall the Marabou-Win tool had to be executed 165 times.

The result of the verification is shown in the result column of the *VerificationTasks* table. Moreover, the column *result_comment* gives more information on the last console output of executing the verification tool. If the tool prints *unsat* in a line the property is verified while a line saying *sat* means that a counter example was found. If none of both was printed the tool did not finish. Exemplary last parts of outputs are shown in section B.1. Here, the timeout for solving each tasks was set to 150 seconds. Having this timeout for 165 execution, having an additional overhead through Cameo, and using an rather slow tool because of hardware restrictions, let to an overall execution time of the script for more than a day. Still, for many verification tasks the tool did not finish in that time and therefore the result is set to unknown. The rest of the verification task were not executed and therefore also got none as result.

5.1.7 *Mapping Results to Requirements*

Afterwards, the opaque behavior *MapResultsOfVTasksOnReqs* is started. The result can be seen in the column *verified* of the *SystemRequirements* table like shown in Figure 5.6. Because the execution of many verification task was interrupted after a timeout of 150 seconds and many properties were not tested, no requirement could be verified yet. However, it was shown that the requirement *NoUnnecessaryManeuvers* is not satisfied. For the other requirements no counter example was found and for some networks the properties hold but there are no guarantees that they hold for all tasks.



#	△ Name	Text	Verified
1	10 DataRequirements	Requirements on the data used for training, validation, and testing	
2	11 RequirementsOnPreprocessing	Requirements on pre-processing like bringing the data into the right format	
3	12 RequirementsOnPostprocessin	Requirements on post-processing steps like calculations later done on the output of the ML model	
4	13 TrainedModelRequirements	Requirements on the trained ML model	
5	13.1 Functional Requirements	Application-specific functional requirements	
6	13.1.6 NoUnnecessaryManeu	Do not advice unnecessary or unnecessary strong behavior if, there is no danger of collision or the intruder is very far away	☐ false
7	13.2 Robustness Requirements	Robustness requirements	
8	13.2.1 NN_Robustness	All trained NNs should be robust	■ <undefi...
9	13.3 Generalization Requireme	Generalization requirements	
10	13.3.1 NN_Generalization	All trained NNs should generalize well	■ <undefi...
11	13.4 Safety Requirements	Application-specific safety requirements	
12	13.4.5 AvoidFrontalCollision	Get out of the way if an intruder is approaching you frontally	■ <undefi...
13	13.4.6 AvoidBumpingIntoIntr	Do not bump into a slower intruder in front of you	■ <undefi...
14	13.4.7 SwerveInRightDirecti	Get out of the way of the intruder in the right direction	■ <undefi...
15	13.5 Fairness Requirements	No sensitive attributes	
16	13.6 Equivalence Requirements	No equivalence checking	
17	14 InferenceModelRequirements	Requirements on the implementation of the ML model on the end-system hardware	

Figure 5.6: Table of System Requirements with Verification Results

5.2 THREAT LOCALIZATION

The aim of the threat localization system proposed by Sprockhoff, Lukic, Janson, et al. [Spr+23] is to detect intruder aircrafts or other threats in the field of view of the camera data of an airplane and to locate those threats. They implemented the simulation of the camera view with the tool Flightgear [OMT], running on a Windows computer while the object detection itself runs on a Jetson Xavier.

5.2.1 *Introduction to the threat Localization System*

The views of camera on the right and left side of the ownship is simulated with Flightgear. On both views object detection is executed to find intruding aircrafts. For detecting, Sprockhoff, Lukic, Janson, et al. [Spr+23] use the popular object detection model YOLOv7 [WBL22]. YOLO is a family of object detection models that have a rather small amount of parameters compared to their performance. They can detect an arbitrary number of objects in an image, return the class of the object, the probability for that it is really that object, and a bounding box around the object. Their architecture is based on a backbone and a head part. However, the number and kinds of layers and activation functions differ between the models. YOLOv7 was published in 2022 and also consist of different models. In the threat localization system the standard YOLOv7 model was used but there exist also a tiny one and extended YOLO models. [WBL22; Sol; Boe]

Based on the bounding boxes that are returned for both camera views and the distance between the left and right camera, the position of the other aircraft or object is calculated.

5.2.2 *Modeling the System Structure*

Firstly, the system structure is specified in a diagram as shown in Figure 5.7 and specify the neural network YOLO which can be seen in Figure 5.8. This is our single object that should be checked here because only requirements on trained models are considered here. The standard YOLO model - which was used by Sprockhoff, Lukic, Janson, et al. [Spr+23] - has 36.9M parameters. It uses the SiLU activation function and mainly consists of re-parameterized convolutional layers with identity connection (RepConvN) and MaxPool layers. [WBL22; Sol]

5.2.3 *Defining the System Requirements*

The formal definition of requirements on image-based ML systems is a challenge, as it is hard to describe properties on the input space. Still, a number of image-specific robustness requirements can be defined to ensure that simple transformations of images do not lead to another prediction. Such transformations can, for example, include translations, rotations, and subsampling. Moreover, robustness against

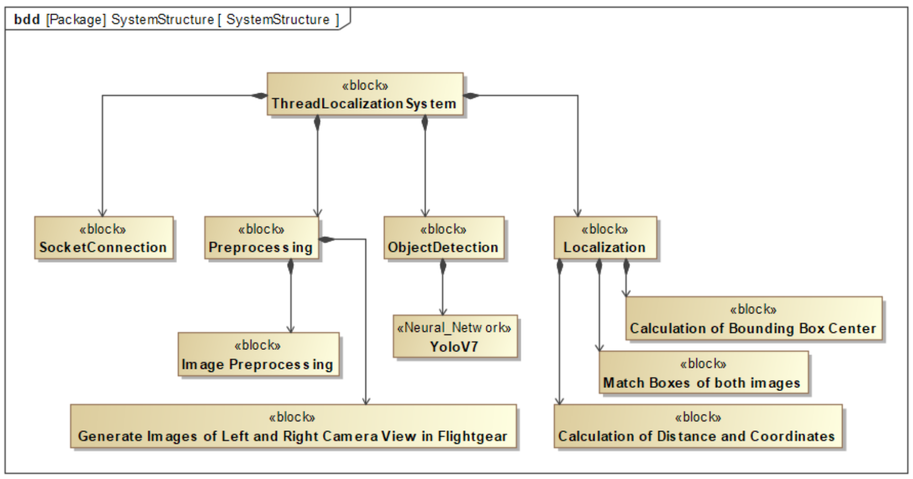
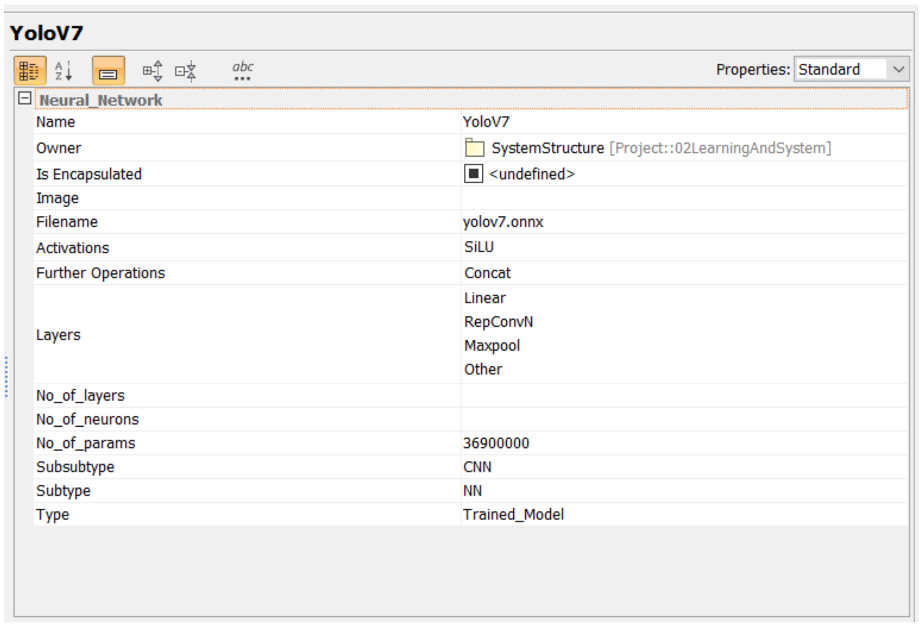

bdd [Package] SystemStructure [SystemStructure]
«block» ThreadLocalizationSystem
«block» SocketConnection
«block» Preprocessing
«block» ObjectDetection
«block» Localization
«block» Image Preprocessing
«Neural_Network» YoloV7
«block» Calculation of Bounding Box Center
«block» Match Boxes of both images
«block» Generate Images of Left and Right Camera View in Flightgear
«block» Calculation of Distance and Coordinates

Figure 5.7: System Structure of the threat localization system


YoloV7

Properties: Standard

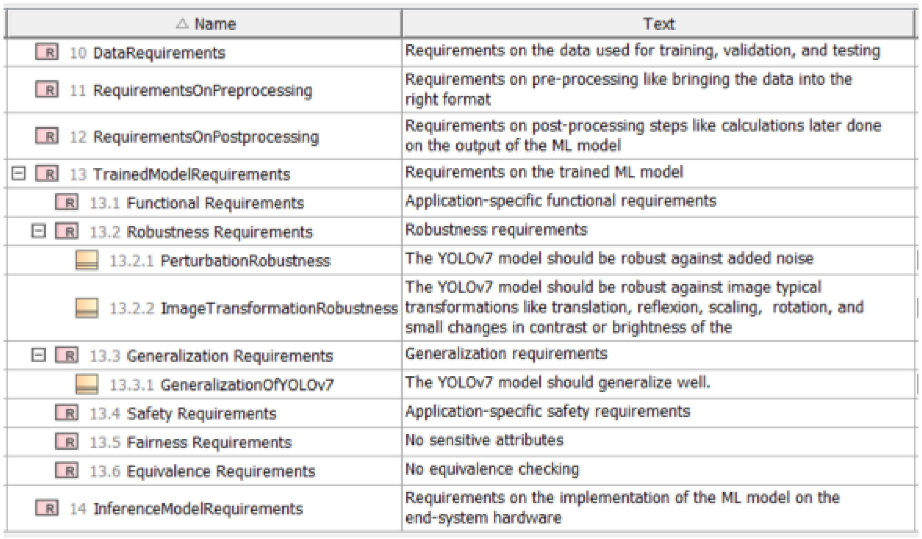
Neural_Network

Name	YoloV7
Owner	SystemStructure [Project::02LearningAndSystem]
Is Encapsulated	<undefined>
Image	
Filename	yolov7.onnx
Activations	SiLU
Further Operations	Concat
Layers	Linear RepConvN Maxpool Other
No_of_layers	
No_of_neurons	
No_of_params	36900000
Subsubtype	CNN
Subtype	NN
Type	Trained_Model

Figure 5.8: The YOLO object

perturbations can be defined for images as input as well. Images that only differ in a small amount of pixels can thereby also be checked to always get the same output.

An overview of the requirements that were defined here for the threat localization is provided in Figure 5.9.



△ Name	Text
10 DataRequirements	Requirements on the data used for training, validation, and testing
11 RequirementsOnPreprocessing	Requirements on pre-processing like bringing the data into the right format
12 RequirementsOnPostprocessing	Requirements on post-processing steps like calculations later done on the output of the ML model
13 TrainedModelRequirements	Requirements on the trained ML model
13.1 Functional Requirements	Application-specific functional requirements
13.2 Robustness Requirements	Robustness requirements
13.2.1 PerturbationRobustness	The YOLOv7 model should be robust against added noise
13.2.2 ImageTransformationRobustness	The YOLOv7 model should be robust against image typical transformations like translation, reflexion, scaling, rotation, and small changes in contrast or brightness of the
13.3 Generalization Requirements	Generalization requirements
13.3.1 GeneralizationOfYOLOv7	The YOLOv7 model should generalize well.
13.4 Safety Requirements	Application-specific safety requirements
13.5 Fairness Requirements	No sensitive attributes
13.6 Equivalence Requirements	No equivalence checking
14 InferenceModelRequirements	Requirements on the implementation of the ML model on the end-system hardware

Figure 5.9: Requirements on the threat localization system

5.2.4 *Defining the Technical Requirements*

Here, this use case has the same technical restriction like in HCAS in subsection 5.1.4. There is no Nvidia-GPU on the laptop and no solver licenses are available.

5.2.5 *Deriving the Properties to Verify*

For each system requirement one or more properties were derived as shown in Figure 5.10.

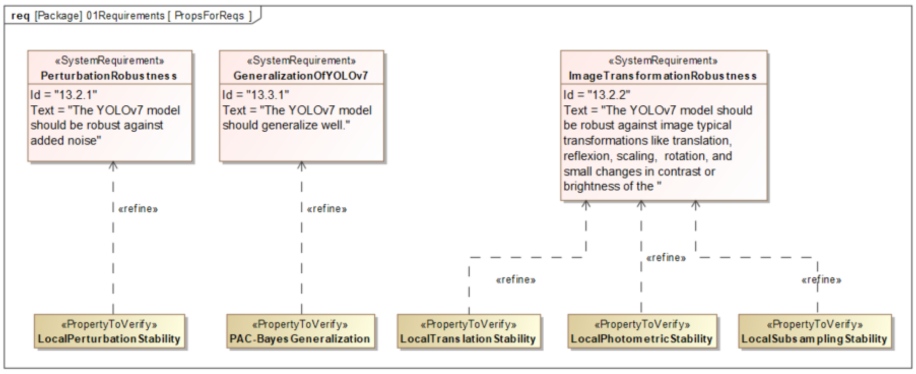

req [Package] 01Requirements [PropsForReqs]
«SystemRequirement» PerturbationRobustness
Id = "13.2.1"
Text = "The YOLOv7 model should be robust against added noise"
«SystemRequirement» GeneralizationOfYOLOv7
Id = "13.3.1"
Text = "The YOLOv7 model should generalize well."
«SystemRequirement» ImageTransformationRobustness
Id = "13.2.2"
Text = "The YOLOv7 model should be robust against image typical transformations like translation, reflexion, scaling, rotation, and small changes in contrast or brightness of the "
«refine»
«refine»
«refine»
«refine»
«refine»
«PropertyToVerify» LocalPerturbationStability
«PropertyToVerify» PAC-Bayes Generalization
«PropertyToVerify» LocalTranslationStability
«PropertyToVerify» LocalPhotometricStability
«PropertyToVerify» LocalSubsamplingStability

Figure 5.10: Refinement of the requirements on the threat localization system

The robustness against perturbations of the pixels is defined through local stability which can be encoded as local input-output property. In addition, the generalization abilities are again measured by PAC-Bayes bound. Therefore, it is a probabilistic property. Furthermore, the requirement on robustness against image-specific transformations is refined by three properties: translation stability, photometric transformation stability, and subsampling stability. Local linear encoding for such properties were proposed by Kouvaros and Lomuscio [KL18]. The definitions still have to be edited for the bounding box output but already show that encoding the hard part, the constraint on the input, is possible.

The table that contains all properties that should be verified including their domains and formal classes is pictured in Figure B.12. All properties should be verified on the YOLO model.

5.2.6 *Determining the Methods and Verification Tasks*

When the script for determining fitting methods is executed no such methods are found. The output of the function *determineMethod()* for one example verification task is shown in section B.2. YOLOv7 uses special kinds of layers and activation functions that are at least not explicitly mentioned to be support by the tools in the table. However, while there is a lot of research on formal verification for classification problems, object detection problems are not handled yet.

5.2.7 *Mapping Results to Requirements*

Therefore, mapping back the verification tasks result to the requirements would only lead to the result none and this step can be skipped. The problem that the YOLO model addresses and its architecture are too complicated and the model has too many special components that - to the best of my knowledge - no formal method is available to verify it. Simulation and scenario-based testing has to be used at the moment instead.

5.3 CONCLUSIONS ON APPLYING THE FRAMEWORK TO THE USE CASES

Applying the framework to the use cases showed that the proposed framework is very useful, as the verification tools dropped out of the list for many different reasons. Without the framework, a user would have to gather the different definitions and approaches and check all those tool characteristics manually. Because HCAS consists of rather small NNs, the framework showed that tools were sorted out because of the technical restrictions on the verification process and because of the property that should be verified. The threat localization system has the complex YOLO model as NN and therefore tools were mainly sorted out because of the characteristics of the YOLO object.

In addition, the model-based approach turned out to be useful because the correctness properties defined for HCAS had all different sets of NNs as targets. In the Cameo model, the properties could be connected to the objects they should be verified on and the verification tasks could be automatically created based on these links and also be connected to the properties and regarded objects.

Moreover, it turned out that still a lot of research is needed on developing more powerful tools and that the tool table could be enhanced by extensively testing all tools on different kinds of models and thereby filling up unknown cells or getting tighter bounds on the tool's capabilities.

6 RELATED WORK, DISCUSSION AND FURTHER IDEAS

This chapter describes how the framework is connected to related work, what the limits of the proposed version of the framework are, and what further steps have to be done to complete the framework for more general scenarios. Moreover, some ideas are discussed that go beyond the basic functionalities.

6.1 RELATED WORK

Related work in terms of the definitions and approaches for the verification of formal properties on neural networks was already given in section 4.5 and section 4.8 because this was the foundation for deriving the framework.

There are also some approaches that build frameworks for the execution of verification tools for local input-output properties. These approaches aim at standardizing input and output formats and making tools comparable. For example, the paper [Liu+20] proposes a standardized implementation of a number of the earlier tools. Further efforts were made by the framework DNNV [SED21] on standardizing the input and output formats of such tools. They suggest using the standard ONNX-format [ONN] for models and specify a format called DNNP for encoding the properties. Then, they converted several benchmarks and tools to support these formats. The competition VNN-COMP [Bak+] also contributes much to the standardization of file input and output formats. The benchmarks used in the competition consist of a model in ONNX-format [ONN] and property in the VNN-LIB [Tac+] format. Furthermore, through the competition report, they provide each year a summary and comparison of state-of-the-art methods in the field. Still, these related works are restricted to local input-output properties and only provide solutions for the verification of a given model and property, without guiding through the selection of properties.

Moreover, the toolkit VerifAI [Dre+19] should be mentioned. It structures the components of the design and verification through simulation of AI components. It defines the necessary inputs for the verification and possible outputs like counter-examples and fuzz testing traces.

Furthermore, the EASA is working on some closely related projects in the area of defining general objectives, regarding ML-specific verification properties, and assembling existing approaches. Their guideline on trustworthy AI was already summarized in subsection 3.3.3. In addition, they published the report of the CoDANN [EAS] project in 2020. This included a W-shaped learning assurance life cycle, general ideas on the safety assessment process, and a practical use case. Moreover, the EASA also proposed a document on the formal verification of learning-based systems this year. The deliverable of the project Formal Methods use for Learning Assurance (ForMuLA) [EA23] gives some general guidance on formal methods and where they can be used in a learning setting. Further guidance is given by the MLEAP interim technical report [Bel+23] of EASA. It splits the problem into three tasks: Data completeness and representativity, model development, and model evaluation. For each of these tasks they define the background concepts, how it is handled for ML and Deep Learning (DL) in specific, and take a look at some methods and tools.

The idea of using MBSE also for AI-based systems was proposed by [Spr+23]. However, they concentrate on using Model-based System-Theoretic Process Analyses and scenario-based safety assessment to address the challenge of assessing safety while this work focuses on integrating formal verification to give safety guarantees.

6.2 DISCUSSION AND FURTHER IDEAS

Since it turned out that it is very hard to see through all the different definitions and approaches, such a framework is needed for integrating verification into ML-based systems. The model-based approach can help users to be practically guided through the process and keep an overview. It brings us a step closer to applying formal verification on complex ML systems to enable the use of AI in safety-critical applications. However, there are still some steps to do to promote this development.

This thesis mainly proposed the framework in general and only a part of the framework was developed in detail. This allows the demonstration of the flow of the framework and its advantages. But still, details on defining requirements were only provided for trained models, verification properties were only formulated for NNs, and the execution of methods was only tested on one tool with four properties on up to 45 NNs. Additional research has to be made for the other components, their requirements, properties to verify, and verification methods. Once the other components with many different characteristics are integrated, decisions have to be made on storing all tools in one table with many empty cells or splitting the table. Based on the type of the object another table could then be read which would reduce the filtering overhead.

A further aspect that can be discussed is the choice of columns of the tool table. They have to capture characteristics of tools that limit their practicality on different kinds of models and properties. ML models can become very complex nowadays and it is hard to capture their complexity just by a number of nodes or parameters. To capture new arising characteristics, the tool table is easily extendable. Moreover, some results on benchmark problems like the execution time and performance could be integrated into the table to make elements within the matching methods comparable. In addition, the information in the tool table just bounds on what is at least possible. Many more architectures and layers might be possible, and for some tools, only a few things are known. Extensively testing all tools to get tighter bounds and filling up the unknown cells in the tool table could help further. In addition, umbrella terms could be handled more intelligently by some kind of inheritance. For example, a tool that supports all piece-wise linear activation functions has to list all common piece-wise linear functions like ReLU and LeakyReLU at the moment. Information that a ReLU function is also a piece-wise linear function can be encoded in the language specification of Cameo in the future.

Moreover, the state-of-the-art tools for formal verification turned out to be still limited by their scalability, the kinds of architectures and layers, and the problems they can address. A lot of research on verifying local input-output properties for classification problems is done while other properties and problems remain open. Other models than NNs also get much less attention. This framework will gain importance with the development of stronger and more diverse tools.

In safety-critical areas, certification is always playing a big role. Formal verification methods can only ensure that a property holds if the verification algorithm is working correctly. While at the moment anyone can add something to the tool table, it could also be possible that a certification agency provides a tool table that only holds certified tools.

The details in this work only focused on the verification of an already trained model. However, there are also many approaches that ensure that certain properties are never violated during the whole training process. Furthermore, there are upcoming algorithms that repair ML models for which a property is violated [Lan]. Integrating these aspects into the framework would be an interesting further step.

The tool table can also be used independently from the rest of the framework and Cameo. This even opens up the opportunity to use the table the other way around for determining how a model can look if it still should be formally verifiable for a certain kind of property.

7 CONCLUSION

This thesis presents a model-based framework that supports the use of formal verification of ML-based systems and demonstrates its use on a collision avoidance system and a threat localization system. The framework provides a way of modeling the verification process and four basic elements: Guidance on choosing requirements, guidance on defining properties to verify, automated support in determining fitting tools, and automatic execution of the verification tasks. These basic elements were implemented for NNs to show all steps of the verification flow. For the guidance on requirement and property selection, extensive literature research was done on guidelines and formal property definitions of NNs. The results were grouped into objectives and formal classes. Based on these formal classes, formal and stochastic verification approaches, methods, and tools were gathered and a CSV table was generated. This table was designed to support the determination of fitting tools for a given problem and to be easily extendable with new tools or new problem characteristics.

For the implementation, the MBSE tool Cameo was used and a package structure, the stereotypes for the required elements, and supporting opaque behaviors were defined. Such opaque behaviors contain Jython scripts that automatically create dependencies and verification task objects including the attribute that contains all fitting methods based on the object and property definitions, run all possible verification tasks with their selected tools, and map back the results of the verification tasks to the system requirements.

The CSV tool table and the Python script for finding fitting tools can also be used independently of any modeling tool. Using the framework on aviation use cases showed that existing verification tools are still very restricted in their options. For example, when applying the HCAS use case on my hardware, many tools dropped off because of the defined technical restrictions like that there is no Nvidia GPU and Windows as the operating system on the laptop. For the threat localization system use case, no fitting verification method was found by the framework because of the complexity of the YOLO model and the object detection task.

In the end, the limits of the framework were discussed and further steps and ideas were presented. Guidance on requirements on other object types and their specific properties can be included in the framework in the future and extensive verification tool testing can be done to make the tool table and thereby the framework even more powerful.

BIBLIOGRAPHY

[Aga+18] A. Agarwal, A. Beygelzimer, M. Dudík, J. Langford, and H. Wallach. *A Reductions Approach to Fair Classification*. 2018. arXiv: 1803.02453 [cs.LG].

[Alb21] A. Albarghouthi. *Introduction to Neural Network Verification*. 2021. arXiv: 2109.10317 [cs.LG].

[Alb+17] A. Albarghouthi, L. D'Antoni, S. Drews, and A. Nori. "FairSquare: probabilistic verification of program fairness". In: *Proceedings of the ACM on Programming Languages* 1 (Oct. 2017), pp. 1–30. DOI: 10.1145/3133904.

[Alq23] P. Alquier. *User-friendly introduction to PAC-Bayes bounds*. 2023. arXiv: 2110.11216 [stat.ML].

[AMFM] A. Ambrosio, F. Mattiello-Francisco, and E. Martins. "A Independent Software Verification and Validation Process for Space Applications". In: *SpaceOps 2008 Conference*. DOI: 10.2514/6.2008-3517. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2008-3517. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2008-3517.

[Ami+23] G. Amir, O. Maayan, T. Zelazny, G. Katz, and M. Schapira. *Verifying Generalization in Deep Learning*. 2023. arXiv: 2302.05745 [cs.LG].

[Amj] H. Amjad. *Combining model checking and theorem proving*. Technical Report. University of Cambridge.

[Aml+05] N. Amla, X. Du, A. Kuehlmann, R. P. Kurshan, and K. L. McMillan. "An Analysis of SAT-Based Model Checking Techniques in an Industrial Environment". In: *Correct Hardware Design and Verification Methods*. Ed. by D. Borrione and W. Paul. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 254–268. ISBN: 978-3-540-32030-2.

[Aro+18] S. Arora, R. Ge, B. Neyshabur, and Y. Zhang. "Stronger Generalization Bounds for Deep Nets via a Compression Approach". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 254–263. URL: https://proceedings.mlr.press/v80/arora18b.html.

[Bak21] S. Bak. “Nnenum: Verification of ReLU Neural Networks with Optimized Abstraction Refinement”. In: *NASA Formal Methods: 13th International Symposium, NFM 2021, Virtual Event, May 24–28, 2021, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2021, 19–36. ISBN: 978-3-030-76383-1. DOI: 10.1007/978-3-030-76384-8_2. URL: https://doi.org/10.1007/978-3-030-76384-8_2.

[BLJ] S. Bak, C. Liu, and T. Johnson. *2nd International Verification of Neural Networks Competition (VNN-COMP'21)*. URL: {https://sites.google.com/view/vnn2021/home} (visited on 05/15/2023).

[BLJ21] S. Bak, C. Liu, and T. Johnson. *The Second International Verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results*. 2021. arXiv: 2109.00498 [cs.LO].

[Bak+] S. Bak, C. Liu, T. Johnson, C. Brix, and M. Müller. *3rd International Verification of Neural Networks Competition (VNN-COMP'22)*. URL: {https://sites.google.com/view/vnn2022} (visited on 05/15/2023).

[BM21] P. K. Banerjee and G. Montufar. “Information Complexity and Generalization Bounds”. In: *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2021. DOI: 10.1109/isit45174.2021.9517960. URL: https://doi.org/10.1109%2Fisit45174.2021.9517960.

[BZSL19] O. Bastani, X. Zhang, and A. Solar-Lezama. *Probabilistic Verification of Fairness Properties via Concentration*. 2019. arXiv: 1812.02573 [cs.AI].

[Bel+23] M. C. T. Belkacem, A. Ioulalen, S. Ribeiro, N. Rodriguez, and J.-B. Rouffet). *EASA Research – Machine Learning Application Approval (MLEAP) interim technical report*. Horizon Europe research and innovation programme report. European Union Aviation Safety Agency, May 2023.

[Bel+18] R. K. E. Bellamy, K. Dey, M. Hind, et al. *AI Fairness 360: An Extensible Toolkit for Detecting, Understanding, and Mitigating Unwanted Algorithmic Bias*. 2018. arXiv: 1810.01943 [cs.AI].

[Bie+03] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. “Bounded Model Checking”. In: *Advances in Computers* (2003).

[BR22] S. Biswas and H. Rajan. *Fairify: Fairness Verification of Neural Networks*. 2022. arXiv: 2212.06140 [cs.LG].

[BNW] N. Bjorner, L. Nachmanson, and C. Wintersteiger. *Z3. An efficient SMT solver*. URL: https://www.microsoft.com/en-us/research/project/z3-3/ (visited on 06/03/2023).

[Boe] G. Boesch. *YOLOv7: The Most Powerful Object Detection Algorithm (2023 Guide)*. URL: https://viso.ai/deep-learning/yolov7-guide/ (visited on 09/27/2023).

[BP] V. Bondalakunta and P. Prabhakar. *AVeriNN*. URL: https://github.com/vishnuteja97/AVeriNN (visited on 05/15/2023).

[BUa] M. Brakel and R. Uuk. *The Act*. URL: https://artificialintelligenceact.eu/the-act/ (visited on 03/17/2023).

[BUb] M. Brakel and R. Uuk. *The Artificial Intelligence Act*. URL: https://artificialintelligenceact.eu/ (visited on 03/17/2023).

[Bri+23] C. Brix, M. N. Müller, S. Bak, T. T. Johnson, and C. Liu. *First Three Years of the International Verification of Neural Networks Competition (VNN-COMP)*. 2023. arXiv: 2301.05815 [cs.LG].

[CW17] N. Carlini and D. Wagner. "Towards Evaluating the Robustness of Neural Networks". In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, pp. 39–57. DOI: 10.1109/SP.2017.49.

[CH23] S. Caton and C. Haas. "Fairness in Machine Learning: A Survey". In: *ACM Comput. Surv.* (2023). Just Accepted. ISSN: 0360-0300. DOI: 10.1145/3616865. URL: https://doi.org/10.1145/3616865.

[Che+23] Z. Chen, J. M. Zhang, F. Sarro, and M. Harman. *An Empirical Study on Fairness Improvement with Multiple Protected Attributes*. 2023. arXiv: 2308.01923 [cs.LG].

[CNR17] C.-H. Cheng, G. Nührenberg, and H. Ruess. "Maximum Resilience of Artificial Neural Networks". In: (Apr. 2017).

[Che+20] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. *A Survey of Model Compression and Acceleration for Deep Neural Networks*. 2020. arXiv: 1710.09282 [cs.LG].

[Dai+21] H. Dai, B. Landry, L. Yang, M. Pavone, and R. Tedrake. *Lyapunov-stable neural-network control*. 2021. arXiv: 2109.14152 [cs.RO].

[Das] Dassault Systems. *CAMEO SYSTEMS MODELER*. URL: https://www.3ds.com/products-services/catia/products/no-magic/cameo-systems-modeler/ (visited on 09/20/2023).

[DS] B. DenBleyker and J. Smith. *VERAPAK*. URL: https://github.com/formal-verification-research/VERAPAK (visited on 06/03/2023).

[Dia+] D. Diamond, D. Weisberger, L. Baker, M. Ward, V. Murali, and J. Naso. *audit-AI. Open Sourced Bias Testing for Generalized Machine Learning Applications*. URL: {https://github.com/pymetrics/audit-ai} (visited on 09/23/2023).

[Dre+19] T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, H. Ravanbakhsh, M. Vazquez-Chanlatte, and S. A. Seshia. "VerifAI: A Toolkit for the Formal Design and Analysis of Artificial Intelligence-Based Systems". In: *31st International Conference on Computer Aided Verification (CAV)*. July 2019.

[EAS] EASA. "Concepts of Design Assurance for Neural Networks (CoDANN). AI Roadmap". In: (). URL: https://www.easa.europa.eu/en/document-library/general-publications/concepts-design-assurance-neural-networks-codann.

[EASa] EASA. "EASA Artificial Intelligence Roadmap 1.0. A human-centric approach to AI in aviation". In: (). URL: https://www.easa.europa.eu/en/downloads/109668/en.

[EASb] EASA. "EASA Concept Paper: First usable guidance for Level 1 and 2 machine learning applications. A deliverable of the EASA AI Roadmap". In: (). URL: https://www.easa.europa.eu/en/downloads/137631/en.

[EASc] EASA. "EASA Concept Paper: First usable guidance for Level 1 machine learning applications. A deliverable of the EASA AI Roadmap". In: (). URL: https://www.easa.europa.eu/en/newsroom-and-events/news/easa-releases-consultation-its-first-usable-guidance-level-1-machine.

[EA23] EASA and C. Aerospace. *Formal Methods use for Learning Assurance (ForMuLA)*. Tech. rep. Apr. 2023.

[Ehl17] R. Ehlers. *Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks*. 2017. arXiv: 1705.01320 [cs.LO].

[Fai] Fair Isaac Corporation. *FICO Xpress Solver. Führende Optimierungs-Solver für lineare, gemischt-ganzzahlige, nicht-lineare oder Constraintprogrammierung.* URL: https://www.fico.com/de/products/fico-xpress-solver (visited on 06/03/2023).

[Fel+15] M. Feldman, S. Friedler, J. Moeller, C. Scheidegger, and S. Venkatasubramanian. *Certifying and removing disparate impact*. 2015. arXiv: 1412.3756 [stat.ML].

[FV19] V. Feldman and J. Vondrak. *Generalization Bounds for Uniformly Stable Algorithms*. 2019. arXiv: 1812.09859 [cs.LG].

[Fer+22] C. Ferrari, M. N. Muller, N. Jovanovic, and M. Vechev. *Complete Verification via Multi-Neuron Relaxation Guided Branch-and-Bound*. 2022. arXiv: 2205.00263 [cs.LG].

[Gan] S. Ganesh. *A Gentle Introduction to Formal Verification*. URL: {https://www.systemverilog.io/verification/gentle-introduction-to-formal-verification/} (visited on 03/14/2023).

[Geh+18] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. "AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation". In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 3–18. DOI: 10.1109/SP.2018.00058.

[Goo] Google AI. *Our Principles. Objectives for AI applications*. URL: https://ai.google/principles (visited on 03/30/2023).

[Gop+18a] D. Gopinath, G. Katz, C. S. Pasareanu, and C. W. Barrett. "DeepSafe: A Data-Driven Approach for Assessing Robustness of Neural Networks". In: *Automated Technology for Verification and Analysis*. 2018. URL: https://api.semanticscholar.org/CorpusID:49430666.

[Gop+18b] D. Gopinath, G. Katz, C. S. Pasăreănu, and C. Barrett. "DeepSafe: A Data-Driven Approach for Assessing Robustness of Neural Networks". In: *Automated Technology for Verification and Analysis*. Ed. by S. K. Lahiri and C. Wang. Cham: Springer International Publishing, 2018, pp. 3–19.

[Gü] K. Gülen. *Round Table: Will there be a global consensus over AI regulation?* URL: https://dataconomy.com/2022/10/artificial-intelligence-laws-and-regulations/ (visited on 03/17/2023).

[HK20] D. J. Hand and S. Khan. “Validating and Verifying AI Systems”. In: *Patterns* 1.3 (2020), p. 100037. ISSN: 2666-3899. DOI: https://doi.org/10.1016/j.patter.2020.100037. URL: https://www.sciencedirect.com/science/article/pii/S2666389920300428.

[HL20] P. Henriksen and A. Lomuscio. “Efficient Neural Network Verification via Adaptive Refinement and Adversarial Search”. In: *European Conference on Artificial Intelligence*. 2020. URL: https://api.semanticscholar.org/CorpusID:208232329.

[Hua+20a] C. Huang, J. Fan, X. Chen, W. Li, and Q. Zhu. “Divide and Slide: Layer-Wise Refinement for Output Range Analysis of Deep Neural Networks”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.11 (2020), pp. 3323–3335. DOI: 10.1109/TCAD.2020.3013071.

[Hua+20b] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi. *A Survey of Safety and Trustworthiness of Deep Neural Networks: Verification, Testing, Adversarial Attack and Defence, and Interpretability*. 2020. arXiv: 1812.08342 [cs.LG].

[IBM] IBM. *IBM ILOG CPLEX Optimization Studio. Build and solve complex optimization models to identify the best possible actions*. URL: https://www.ibm.com/products/ilog-cplex-optimization-studio (visited on 06/03/2023).

[Ind] Independend high-level Expert Group on Artificial Intelligence set up by the European Commission. “Ethics Guidelines for trustworthy AI”. In: (). URL: https://ec.europa.eu/newsroom/dae/document.cfm?doc_id=60419.

[Jia+19] Y. Jiang, B. Neyshabur, H. Mobahi, D. Krishnan, and S. Bengio. *Fantastic Generalization Measures and Where to Find Them*. 2019. arXiv: 1912.02178 [cs.LG].

[Job] A. Jobin. “The global landscape of AI ethics guidelines”. In: *Nature Machine Intelligence* (). URL: https://doi.org/10.1038/s42256-019-0088-2.

[JLZ23] H. Ju, D. Li, and H. R. Zhang. *Robust Fine-Tuning of Deep Neural Networks with Hessian-based Generalization Guarantees*. 2023. arXiv: 2206.02659 [cs.LG].

[Jul+16] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer. "Policy compression for aircraft collision avoidance systems". In: *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)* (2016), pp. 1–10. URL: https://api.semanticscholar.org/CorpusID:3123038.

[Kat+17] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. *Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks*. 2017. arXiv: 1702.01135 [cs.AI].

[Kat+] G. Katz, D. Huang, D. Ibeling, et al. *Marabou*. URL: https://github.com/NeuralNetworkVerification/Marabou (visited on 07/15/2023).

[Kat+19] G. Katz, D. Huang, D. Ibeling, et al. "The Marabou Framework for Verification and Analysis of Deep Neural Networks". In: July 2019, pp. 443–452. ISBN: 978-3-030-25539-8. DOI: 10.1007/978-3-030-25540-4_26.

[KFG] N. Kershaw, T. Fosmark, and C. Gronlund. *Introduction to guidelines for human-AI interaction*. URL: https://learn.microsoft.com/en-us/ai/guidelines-human-ai-interaction/ (visited on 03/30/2023).

[KFS21] H. Khedr, J. Ferlez, and Y. Shoukry. *PEREGRiNN: Penalized-Relaxation Greedy Neural Network Verifier*. 2021. arXiv: 2006.10864 [cs.LG].

[KS23] G. Khromov and S. P. Singh. *Some Fundamental Aspects about Lipschitz Continuity of Neural Network Functions*. 2023. arXiv: 2302.10886 [cs.LG].

[Kir+23] R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel. "A Survey of Zero-shot Generalisation in Deep Reinforcement Learning". In: *Journal of Artificial Intelligence Research* 76 (2023), pp. 201–264. DOI: 10.1613/jair.1.14174. URL: https://doi.org/10.1613%2Fjair.1.14174.

[KBKS20] M. Kleine Büning, P. Kern, and C. Sinz. "Verifying Equivalence Properties of Neural Networks with ReLU Activation Functions". In: *Principles and Practice of Constraint Programming: 26th International Conference, CP 2020, Louvain-La-Neuve, Belgium, September 7–11, 2020, Proceedings*. Louvain-la-Neuve, Belgium: Springer-Verlag, 2020, 868–884. ISBN: 978-3-030-58474-0. DOI: 10.1007/978-3-030-58475-7_50. URL: https://doi.org/10.1007/978-3-030-58475-7_50.

[KL18] P. Kouvaros and A. Lomuscio. *Formal Verification of CNN-based Perception Systems*. 2018. arXiv: 1811.11373 [cs.LG].

[Kri+18] Krishnamurthy, Dvijotham, R. Stanforth, S. Gowal, T. Mann, and P. Kohli. *A Dual Approach to Scalable Verification of Deep Networks*. 2018. arXiv: 1803.06567 [cs.LG].

[Lan] N. Landon. "A survey of repair strategies for deep neural networks". Master's thesis. Iowa State University. URL: https://dr.lib.iastate.edu/entities/publication/72247a81-84a0-4b75-b138-86eb32516d8f/full (visited on 06/14/2023).

[LK22] N. Levy and G. Katz. *RoMA: a Method for Neural Network Robustness Measurement and Assessment*. 2022. arXiv: 2110.11088 [cs.LG].

[Liu+20] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, and M. J. Kochenderfer. *Algorithms for Verifying Deep Neural Networks*. 2020. arXiv: 1903.06758 [cs.LG].

[LJ] C. Liu and T. Johnson. *VNN-COMP*. URL: {https://sites.google.com/view/vnn20/vnncomp} (visited on 05/15/2023).

[MNO19] R. Mangal, A. V. Nori, and A. Orso. *Robustness of Neural Networks: A Probabilistic and Practical Approach*. 2019. arXiv: 1902.05983 [cs.LG].

[Men+22] M. H. Meng, G. Bai, S. G. Teo, Z. Hou, Y. Xiao, Y. Lin, and J. S. Dong. "Adversarial Robustness of Deep Neural Networks: A Survey from a Formal Verification Perspective". In: *IEEE Transactions on Dependable and Secure Computing* (2022). DOI: 10.1109/TDSC.2022.3179131.

[MRP] A. G. Mukul R. Prasad Armin Biere. "A survey of recent advances in SAT-based formal verification". In: *International Journal on Software Tools for Technology Transfer* ().

[Mü+23] M. N. Müller, C. Brix, S. Bak, C. Liu, and T. T. Johnson. *The Third International Verification of Neural Networks Competition (VNN-COMP 2022): Summary and Results*. 2023. arXiv: 2212.10376 [cs.LG].

[NAS04] NASA. "NASA Framework for the Ethical Use of Artificial Intelligence (AI)". In: (4–2021). URL: https://ntrs.nasa.gov/api/citations/20210012886/downloads/NASA-TM-20210012886.pdf.

[NIS01] NIST. "Artificial Intelligence Risk Management Framework (AI RMF 1.0)". In: (1–2023). URL: https://doi.org/10.6028/NIST.AI.100-1.

[Nag+23] V. Nagisetty, L. Graves, G. Pan, P. Jha, and V. Ganesh. *CGDTest: A Constrained Gradient Descent Algorithm for Testing Neural Networks*. 2023. arXiv: 2304.01826 [cs.LG].

[Nil] N. J. Nilsson. *INTRODUCTION TO MACHINE LEARNING. AN EARLY DRAFT OF A PROPOSED TEXTBOOK.*

[Nvi] Nvidia. *CUDA Toolkit. Develop, Optimize and Deploy GPU-Accelerated Apps*. URL: {https://developer.nvidia.com/cuda-toolkit} (visited on 09/23/2023).

[ONN] ONNX-Community. *Open Neural Network Exchange. The open standard for machine learning interoperability*. URL: {https://onnx.ai/} (visited on 09/14/2023).

[OMT] C. L. Olson, T. Moore, and J. Turner. *FlightGear Flight Simulator. sophisticated, professional, open-source*. URL: https://www.flightgear.org/ (visited on 10/06/2023).

[PGA23] P. Pauli, D. Gramlich, and F. Allgöwer. *Lipschitz constant estimation for 1D convolutional neural networks*. 2023. arXiv: 2211.15253 [cs.LG].

[Pau+22a] P. Pauli, A. Koch, J. Berberich, P. Kohler, and F. Allgower. "Training Robust Neural Networks Using Lipschitz Bounds". In: *IEEE Control Systems Letters* 6 (2022), pp. 121–126. DOI: 10.1109/lcsys.2021.3050444. URL: https://doi.org/10.1109%2Flcsys.2021.3050444.

[PWW20] B. Paulsen, J. Wang, and C. Wang. "ReluDiff". In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. ACM, 2020. DOI: 10.1145/3377811.3380337. URL: https://doi.org/10.1145%2F3377811.3380337.

[Pau+22b] M. Pautov, N. Tursynbek, M. Munkhoeva, N. Muravev, A. Petiushko, and I. Oseledets. "CC-CERT: A Probabilistic Approach to Certify General Robustness of Neural Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.7 (2022), pp. 7975–7983. DOI: 10.1609/aaai.v36i7.20768. URL: https://ojs.aaai.org/index.php/AAAI/article/view/20768.

[RSL20] A. Raghunathan, J. Steinhardt, and P. Liang. *Certified Defenses against Adversarial Examples*. 2020. arXiv: 1801.09344 [cs.LG].

[RYH22] D. A. Roberts, S. Yaida, and B. Hanin. *The Principles of Deep Learning Theory*. Cambridge University Press, 2022. DOI: 10.1017/9781009023405. URL: https://doi.org/10.1017%2F9781009023405.

[Roc+22] E. A. Rocamora, M. F. Sahin, F. Liu, G. G. Chrysos, and V. Cevher. *Sound and Complete Verification of Polynomial Networks*. 2022. arXiv: 2209.07235 [cs.LG].

[Rua+19] W. Ruan, M. Wu, Y. Sun, X. Huang, D. Kroening, and M. Kwiatkowska. "Global Robustness Evaluation of Deep Neural Networks with Provable Guarantees for the Hamming Distance". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 5944–5952. DOI: 10.24963/ijcai.2019/824. URL: https://doi.org/10.24963/ijcai.2019/824.

[Ryo+21] W. Ryou, J. Chen, M. Balunovic, G. Singh, A. Dan, and M. Vechev. *Fast and Effective Robustness Certification for Recurrent Neural Networks*. 2021. arXiv: 2005.13300 [cs.LG].

[SAE21] SAE International. "Artificial Intelligence in Aeronautical Systems: Statement of Concerns (AIR6988". In: (Apr. 2021). URL: https://www.sae.org/standards/content/air6988/.

[Sal+19] P. Saleiro, B. Kuester, L. Hinkson, J. London, A. Stevens, A. Anisfeld, K. T. Rodolfa, and R. Ghani. *Aequitas: A Bias and Fairness Audit Toolkit*. 2019. arXiv: 1811.05577 [cs.LG].

[Sch02] S. Schwoon. "Model-Checking Pushdown Systems". Dissertation. Technische Universität München, 2002.

[SS16] S. A. Seshia and D. Sadigh. "Towards Verified Artificial Intelligence". In: *CoRR* abs/1606.08514 (2016). arXiv: 1606.08514. URL: http://arxiv.org/abs/1606.08514.

[She20] N. Shevchenko. *An Introduction to Model-Based Systems Engineering (MBSE)*. Carnegie Mellon University, Software Engineering Institute's Insights (blog). Accessed: 2023-Sep-20. 2020. URL: https://insights.sei.cmu.edu/blog/introduction-model-based-systems-engineering-mbse/.

[Shi] Q. J. Shi. *Generalisation Bounds (4): PAC Bayesian Bounds*. The Australian Centre for Visual Technologies, The University of Adelaide, Australia. URL: https://cs.adelaide.edu.au/~javen/talk/bounds_slide4.pdf.

[SED21] D. Shriver, S. Elbaum, and M. B. Dwyer. "DNNV: A Framework for Deep Neural Network Verification". In: *Computer Aided Verification*. Springer International Publishing, 2021, pp. 137–150. DOI: 10.1007/978-3-030-81685-8_6. URL: https://doi.org/10.1007%2F978-3-030-81685-8_6.

[Sio] L. Sioli. *A European Strategy for Artificial Intelligence*. URL: https://www.ceps.eu/wp-content/uploads/2021/04/AI-Presentation-CEPS-Webinar-L.-Sioli-23.4.21.pdf? (visited on 03/17/2023).

[Sol] J. Solawetz. *What is YOLOv7? A Complete Guide*. URL: {https://blog.roboflow.com/yolov7-breakdown/} (visited on 09/27/2023).

[Spa] A. Sparrius. "Everything You Thought You Knew about Validation and Verification is Probably Dodgy". In: *12th INCOSE SA Systems Engineering Conference* (). URL: {https://documents.pub/document/everything-you-thought-you-knew-about-validation-everything-you-thought-you.html?page=1}.

[Spr+23] J. Sprockhoff, B. Lukic, V. Janson, A. Ahlbrecht, U. Durak, S. Gupta, and T. Krueger. "Model-Based Systems Engineering for AI-Based Systems". In: Jan. 2023. DOI: 10.2514/6.2023-2587.

[Tac+] A. Tacchella, L. Pulina, D. Guidotti, and S. Demarchi. *VNN-LIB. The international benchmarks standard for the Verification of Neural Networks*. URL: {https://www.vnnlib.org/} (visited on 09/14/2023).

[Tea] C. Team. *The Machine Learning Process. Learn the general structure of how to approach Machine Learning problems in a methodical way.* URL: https://www.codecademy.com/article/the-ml-process (visited on 09/20/2023).

[Teu+21] S. Teuber, M. K. Büning, P. Kern, and C. Sinz. *Geometric Path Enumeration for Equivalence Verification of Neural Networks*. 2021. arXiv: 2112.06582 [cs.LG].

[The20] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[TXT19] V. Tjeng, K. Xiao, and R. Tedrake. *Evaluating Robustness of Neural Networks with Mixed Integer Programming*. 2019. arXiv: 1711.07356 [cs.LG].

[TBB] E. Towle, S. Bowly, and I. Burgesse. *GUROBI OPTIMIZATION. Solve Complex Problems, Fast*. URL: https://www.gurobi.com/ (visited on 06/03/2023).

[Tra+16] F. Tramer, V. Atlidakis, R. Geambasu, D. Hsu, J.-P. Hubaux, M. Humbert, A. Juels, and H. Lin. *FairTest: Discovering Unwarranted Associations in Data-Driven Applications*. 2016. arXiv: 1510.02377 [cs.CY].

[Tra+20] H.-D. Tran, X. Yang, D. M. Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson. *NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems*. 2020. arXiv: 2004.05519 [eess.SY].

[Urb+20] C. Urban, M. Christakis, V. Wüstholz, and F. Zhang. *Perfectly Parallel Fairness Certification of Neural Networks*. 2020. arXiv: 1912.02499 [cs.PL].

[VPL20] G. Valle-Pérez and A. A. Louis. *Generalization bounds for deep learning*. 2020. arXiv: 2012.04115 [stat.ML].

[Var09] M. Vardi. "Model Checking as A Reachability Problem". In: vol. 5797. Sept. 2009, p. 35. ISBN: 978-3-642-04419-9. DOI: 10.1007/978-3-642-04420-5_5.

[WBL22] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. arXiv: 2207.02696 [cs.CV].

[Wan+18a] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. *Efficient Formal Safety Analysis of Neural Networks*. 2018. arXiv: 1809.08098 [cs.LG].

[Wan+18b] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. *Formal Security Analysis of Neural Networks using Symbolic Intervals*. 2018. arXiv: 1804.10829 [cs.AI].

[Wan+21] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter. "Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification". In: *Advances in Neural Information Processing Systems* 34 (2021).

[WHZ22] Z. Wang, C. Huang, and Q. Zhu. *Efficient Global Robustness Certification of Neural Networks via Interleaving Twin-Network Encoding*. 2022. arXiv: 2203.14141 [cs.LG].

[Web+19] S. Webb, T. Rainforth, Y. W. Teh, and M. P. Kumar. *A Statistical Approach to Assessing Neural Network Robustness*. 2019. arXiv: 1811.07209 [stat.ML].

[Wen+18] L. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, L. Daniel, D. Boning, and I. Dhillon. “Towards Fast Computation of Certified Robustness for ReLU Networks”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 5276–5285. URL: https://proceedings.mlr.press/v80/weng18a.html.

[Wen+19] T.-W. Weng, P.-Y. Chen, L. M. Nguyen, M. S. Squillante, I. Oseledets, and L. Daniel. *PROVEN: Certifying Robustness of Neural Networks with a Probabilistic Approach*. 2019. arXiv: 1812.08329 [cs.LG].

[XTJ17] W. Xiang, H.-D. Tran, and T. T. Johnson. *Reachable Set Computation and Safety Verification for Neural Networks with ReLU Activations*. 2017. arXiv: 1712.08163 [cs.LG].

[XTJ18] W. Xiang, H.-D. Tran, and T. T. Johnson. *Output Reachable Set Estimation and Verification for Multi-Layer Neural Networks*. 2018. arXiv: 1708.03322 [cs.LG].

[Xu+21] K. Xu, H. Zhang, S. Wang, Y. Wang, S. Jana, X. Lin, and C.-J. Hsieh. *Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers*. 2021. arXiv: 2011.13824 [cs.AI].

[Xue+22] A. Xue, L. Lindemann, A. Robey, H. Hassani, G. J. Pappas, and R. Alur. *Chordal Sparsity for Lipschitz Constant Estimation of Deep Neural Networks*. 2022. arXiv: 2204.00846 [cs.LG].

[Yua+18] X. Yuan, P. He, Q. Zhu, and X. Li. *Adversarial Examples: Attacks and Defenses for Deep Learning*. 2018. arXiv: 1712.07107 [cs.LG].

[Zha+22a] B. Zhang, D. Jiang, D. He, and L. Wang. *Rethinking Lipschitz Neural Networks and Certified Robustness: A Boolean Function Perspective*. 2022. arXiv: 2210.01787 [cs.LG].

[Zha+21] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. “Understanding Deep Learning (Still) Requires Rethinking Generalization”. In: *Commun. ACM* 64.3 (2021), 107–115. ISSN: 0001-0782. DOI: 10.1145/3446776. URL: https://doi.org/10.1145/3446776.

[Zha+22b] H. Zhang, S. Wang, K. Xu, L. Li, B. Li, S. Jana, C.-J. Hsieh, and J. Z. Kolter. *General Cutting Planes for Bound-Propagation-Based Neural Network Verification*. 2022. arXiv: 2208.05740 [cs.LG].

[Zha+22c] H. Zhang, S. Wang, K. Xu, Y. Wang, S. Jana, C.-J. Hsieh, and Z. Kolter. “A Branch and Bound Framework for Stronger Adversarial Attacks of ReLU Networks”. In: *Proceedings of the 39th International Conference on Machine Learning*. Vol. 162. 2022, pp. 26591–26604.

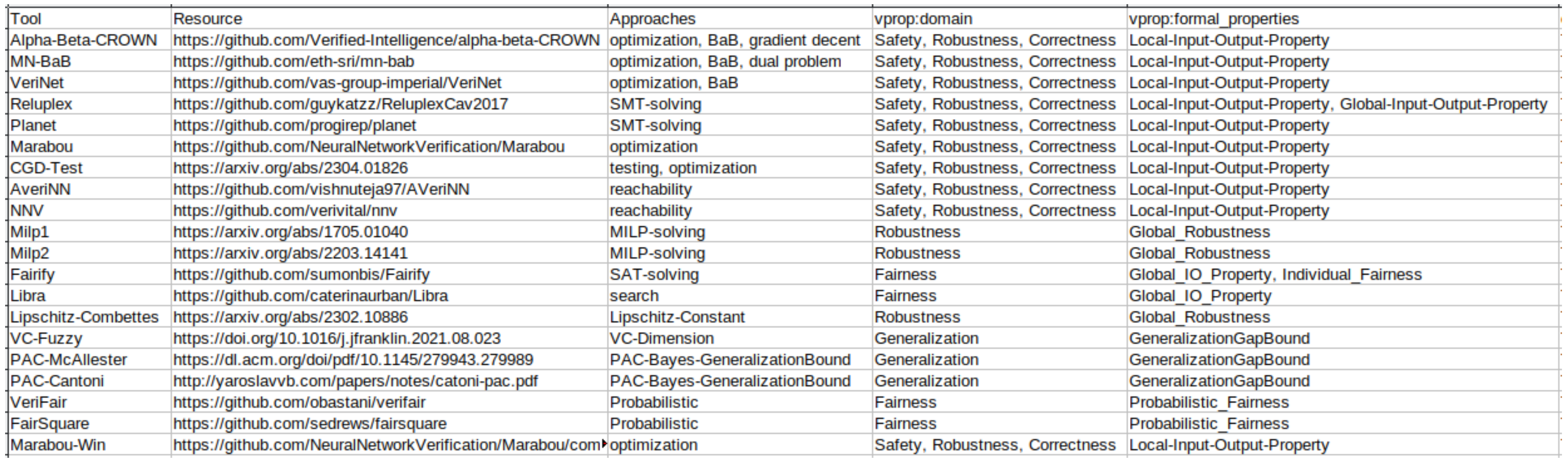
[Zha+18] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel. *Efficient Neural Network Robustness Certification with General Activation Functions*. 2018. arXiv: `1811.00866 [cs.LG]`.

[Zha+] H. Zhang, K. Xu, Z. Shi, J. Chen, L. Li, S. Wang, Z. Yang, and Y. Wang. *alpha-beta-CROWN: A Fast and Scalable Neural Network Verifier using the Bound Propagation Framework*. URL: `https://github.com/Verified-Intelligence/alpha-beta-CROWN` (visited on 05/15/2023).

[ZS22] M. Zhang and J. Sun. *Adaptive Fairness Improvement Based on Causality Analysis*. 2022. arXiv: `2209.07190 [cs.LG]`.

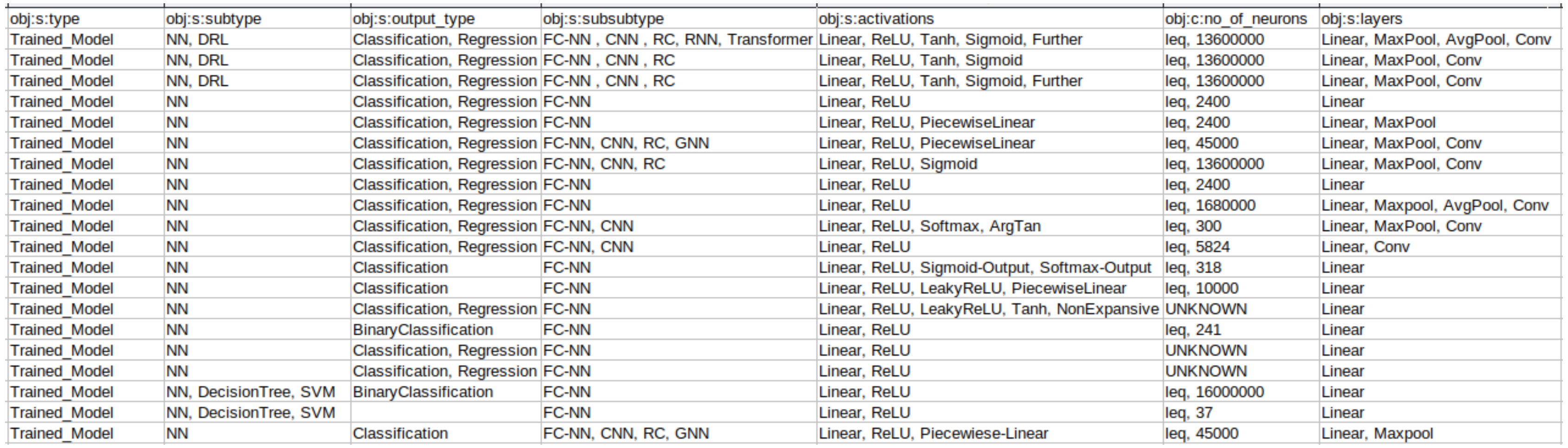
A
TOOL TABLE

The figure of the tool table was split vertically in three parts because it has too many columns with tool characteristic to fit on one page. These parts can be found in Figure A.1, Figure A.2 , and Figure A.3.



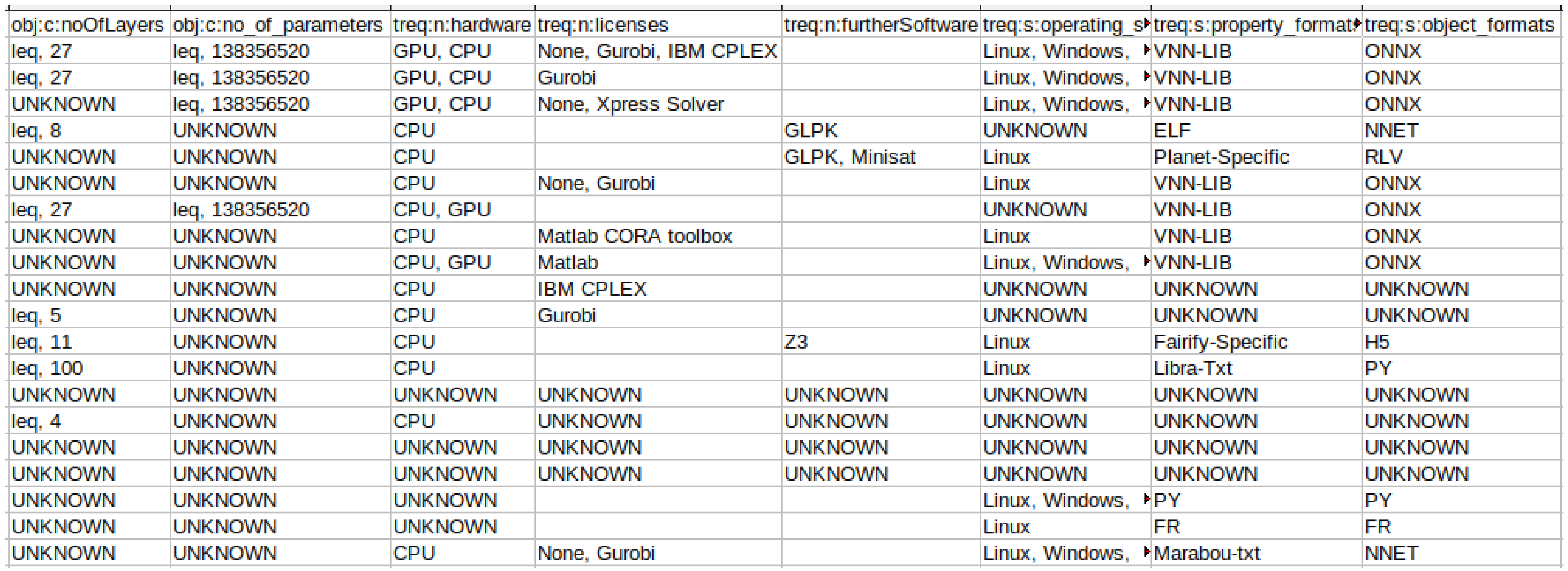
Tool	Resource	Approaches	vprop:domain	vprop:formal_properties
Alpha-Beta-CROWN	https://github.com/Verified-Intelligence/alpha-beta-CROWN	optimization, BaB, gradient decent	Safety, Robustness, Correctness	Local-Input-Output-Property
MN-BaB	https://github.com/eth-sri/mn-bab	optimization, BaB, dual problem	Safety, Robustness, Correctness	Local-Input-Output-Property
VeriNet	https://github.com/vas-group-imperial/VeriNet	optimization, BaB	Safety, Robustness, Correctness	Local-Input-Output-Property
Reluplex	https://github.com/guykatzz/ReluplexCav2017	SMT-solving	Safety, Robustness, Correctness	Local-Input-Output-Property, Global-Input-Output-Property
Planet	https://github.com/progirep/planet	SMT-solving	Safety, Robustness, Correctness	Local-Input-Output-Property
Marabou	https://github.com/NeuralNetworkVerification/Marabou	optimization	Safety, Robustness, Correctness	Local-Input-Output-Property
CGD-Test	https://arxiv.org/abs/2304.01826	testing, optimization	Safety, Robustness, Correctness	Local-Input-Output-Property
AveriNN	https://github.com/vishnuteja97/AVeriNN	reachability	Safety, Robustness, Correctness	Local-Input-Output-Property
NNV	https://github.com/verivital/nnv	reachability	Safety, Robustness, Correctness	Local-Input-Output-Property
Milp1	https://arxiv.org/abs/1705.01040	MILP-solving	Robustness	Global_Robustness
Milp2	https://arxiv.org/abs/2203.14141	MILP-solving	Robustness	Global_Robustness
Fairify	https://github.com/sumonbis/Fairify	SAT-solving	Fairness	Global_IO_Property, Individual_Fairness
Libra	https://github.com/caterinaurban/Libra	search	Fairness	Global_IO_Property
Lipschitz-Combettes	https://arxiv.org/abs/2302.10886	Lipschitz-Constant	Robustness	Global_Robustness
VC-Fuzzy	https://doi.org/10.1016/j.jfranklin.2021.08.023	VC-Dimension	Generalization	GeneralizationGapBound
PAC-McAllester	https://dl.acm.org/doi/pdf/10.1145/279943.279989	PAC-Bayes-GeneralizationBound	Generalization	GeneralizationGapBound
PAC-Cantoni	http://yaroslavvb.com/papers/notes/catoni-pac.pdf	PAC-Bayes-GeneralizationBound	Generalization	GeneralizationGapBound
VeriFair	https://github.com/obastani/verifair	Probabilistic	Fairness	Probabilistic_Fairness
FairSquare	https://github.com/sedrews/fairsquare	Probabilistic	Fairness	Probabilistic_Fairness
Marabou-Win	https://github.com/NeuralNetworkVerification/Marabou/com	optimization	Safety, Robustness, Correctness	Local-Input-Output-Property

Figure A.1: Tool Table Part 1



obj:s:type	obj:s:subtype	obj:s:output_type	obj:s:subsubtype	obj:s:activations	obj:c:no_of_neurons	obj:s:layers
Trained_Model	NN, DRL	Classification, Regression	FC-NN , CNN , RC, RNN, Transformer	Linear, ReLU, Tanh, Sigmoid, Further	leq, 13600000	Linear, MaxPool, AvgPool, Conv
Trained_Model	NN, DRL	Classification, Regression	FC-NN , CNN , RC	Linear, ReLU, Tanh, Sigmoid	leq, 13600000	Linear, MaxPool, Conv
Trained_Model	NN, DRL	Classification, Regression	FC-NN , CNN , RC	Linear, ReLU, Tanh, Sigmoid, Further	leq, 13600000	Linear, MaxPool, Conv
Trained_Model	NN	Classification, Regression	FC-NN	Linear, ReLU	leq, 2400	Linear
Trained_Model	NN	Classification, Regression	FC-NN	Linear, ReLU, PiecewiseLinear	leq, 2400	Linear, MaxPool
Trained_Model	NN	Classification, Regression	FC-NN, CNN, RC, GNN	Linear, ReLU, PiecewiseLinear	leq, 45000	Linear, MaxPool, Conv
Trained_Model	NN	Classification, Regression	FC-NN, CNN, RC	Linear, ReLU, Sigmoid	leq, 13600000	Linear, MaxPool, Conv
Trained_Model	NN	Classification, Regression	FC-NN	Linear, ReLU	leq, 2400	Linear
Trained_Model	NN	Classification, Regression	FC-NN	Linear, ReLU	leq, 1680000	Linear, Maxpool, AvgPool, Conv
Trained_Model	NN	Classification, Regression	FC-NN, CNN	Linear, ReLU, Softmax, ArgTan	leq, 300	Linear, MaxPool, Conv
Trained_Model	NN	Classification, Regression	FC-NN, CNN	Linear, ReLU	leq, 5824	Linear, Conv
Trained_Model	NN	Classification	FC-NN	Linear, ReLU, Sigmoid-Output, Softmax-Output	leq, 318	Linear
Trained_Model	NN	Classification	FC-NN	Linear, ReLU, LeakyReLU, PiecewiseLinear	leq, 10000	Linear
Trained_Model	NN	Classification, Regression	FC-NN	Linear, ReLU, LeakyReLU, Tanh, NonExpansive	UNKNOWN	Linear
Trained_Model	NN	BinaryClassification	FC-NN	Linear, ReLU	leq, 241	Linear
Trained_Model	NN	Classification, Regression	FC-NN	Linear, ReLU	UNKNOWN	Linear
Trained_Model	NN	Classification, Regression	FC-NN	Linear, ReLU	UNKNOWN	Linear
Trained_Model	NN, DecisionTree, SVM	BinaryClassification	FC-NN	Linear, ReLU	leq, 16000000	Linear
Trained_Model	NN, DecisionTree, SVM		FC-NN	Linear, ReLU	leq, 37	Linear
Trained_Model	NN	Classification	FC-NN, CNN, RC, GNN	Linear, ReLU, Piecewiese-Linear	leq, 45000	Linear, Maxpool

Figure A.2: Tool Table Part 2



obj:c:noOfLayers	obj:c:no_of_parameters	treq:n:hardware	treq:n:licenses	treq:n:furtherSoftware	treq:s:operating_s	treq:s:property_format	treq:s:object_formats
leq, 27	leq, 138356520	GPU, CPU	None, Gurobi, IBM CPLEX		Linux, Windows,	VNN-LIB	ONNX
leq, 27	leq, 138356520	GPU, CPU	Gurobi		Linux, Windows,	VNN-LIB	ONNX
UNKNOWN	leq, 138356520	GPU, CPU	None, Xpress Solver		Linux, Windows,	VNN-LIB	ONNX
leq, 8	UNKNOWN	CPU		GLPK	UNKNOWN	ELF	NNET
UNKNOWN	UNKNOWN	CPU		GLPK, Minisat	Linux	Planet-Specific	RLV
UNKNOWN	UNKNOWN	CPU	None, Gurobi		Linux	VNN-LIB	ONNX
leq, 27	leq, 138356520	CPU, GPU			UNKNOWN	VNN-LIB	ONNX
UNKNOWN	UNKNOWN	CPU	Matlab CORA toolbox		Linux	VNN-LIB	ONNX
UNKNOWN	UNKNOWN	CPU, GPU	Matlab		Linux, Windows,	VNN-LIB	ONNX
UNKNOWN	UNKNOWN	CPU	IBM CPLEX		UNKNOWN	UNKNOWN	UNKNOWN
leq, 5	UNKNOWN	CPU	Gurobi		UNKNOWN	UNKNOWN	UNKNOWN
leq, 11	UNKNOWN	CPU		Z3	Linux	Fairify-Specific	H5
leq, 100	UNKNOWN	CPU			Linux	Libra-Txt	PY
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
leq, 4	UNKNOWN	CPU	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN			Linux, Windows,	PY	PY
UNKNOWN	UNKNOWN	UNKNOWN			Linux	FR	FR
UNKNOWN	UNKNOWN	CPU	None, Gurobi		Linux, Windows,	Marabou-txt	NNET

Figure A.3: Tool Table Part 3

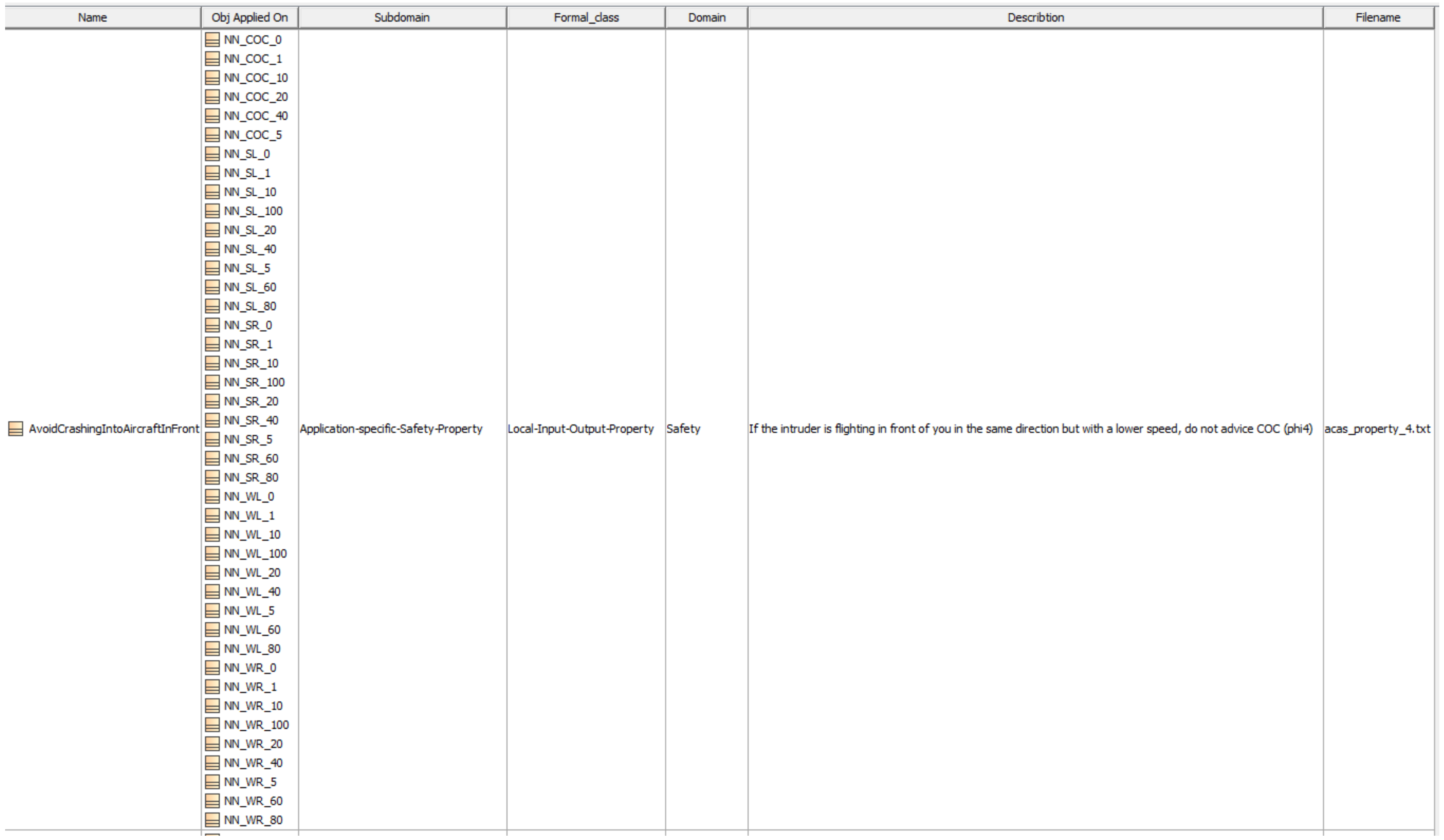
B FURTHER MATERIAL ON THE USE CASES

B.1 FURTHER FIGURES ON APPLYING THE FRAMEWORK FOR HCAS

The complete verification task table was also split and can be found in Figure B.2, Figure B.3, Figure B.4, Figure B.5, Figure B.6, and Figure B.7. Two example outputs of running the *DetermineMethod()* function on a local input-output property and on a generalization property are shown in Figure B.8 and Figure B.9. Figure B.10 and Figure B.11 show examples of the console outputs from running Marabou that were saved in the *VerificationTask* objects.

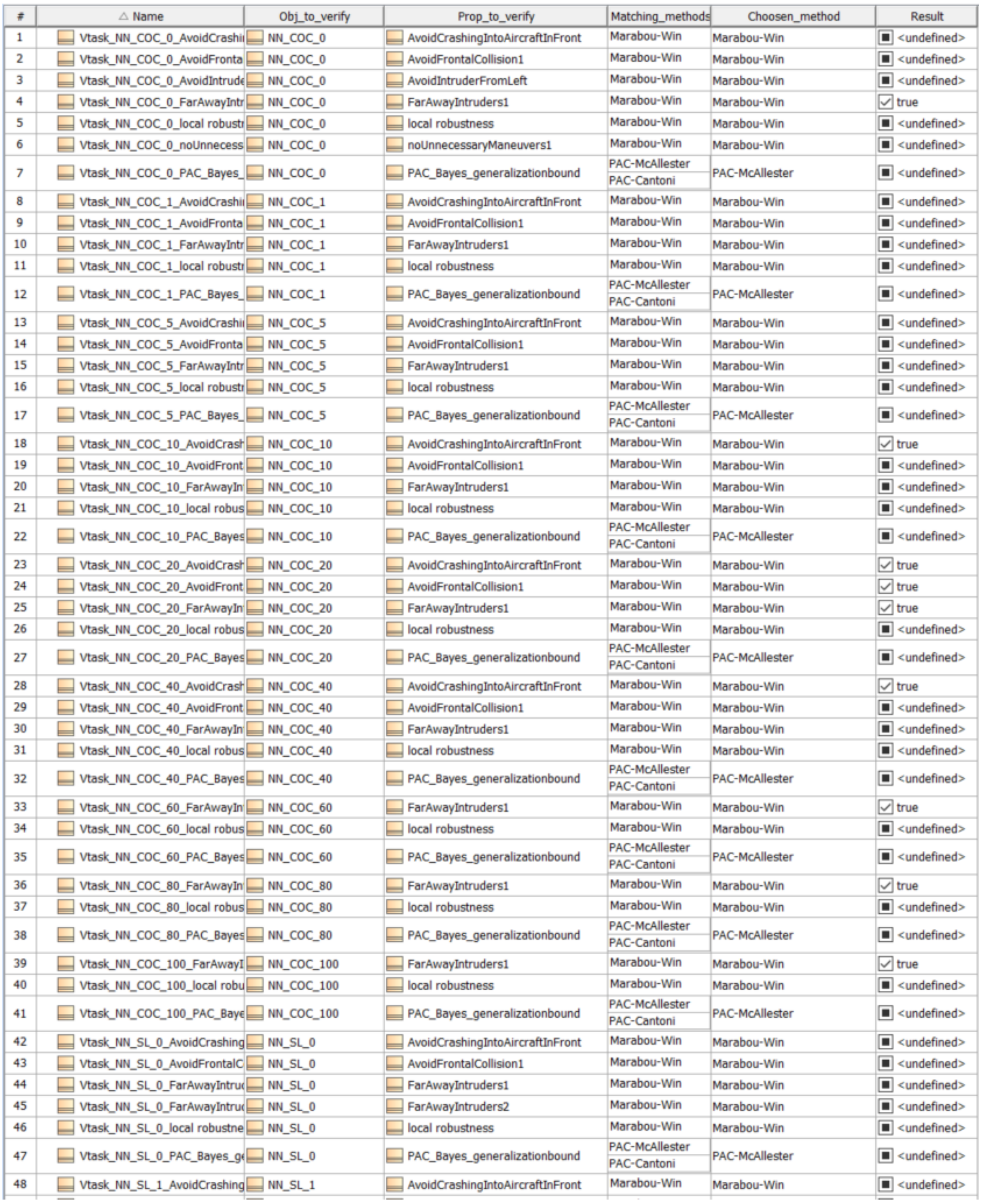
B.2 FURTHER FIGURES ON APPLYING THE FRAMEWORK ON THE THREAT LOCALIZATION SYSTEM

Figure B.13 and Figure B.14 show the console outputs of the *Deter-mineMethod* function. For the first example, the attribute *output_type* was not set and therefore ignored in the selection of matching methods. In the second example, it was set to *ObjectDetection*.



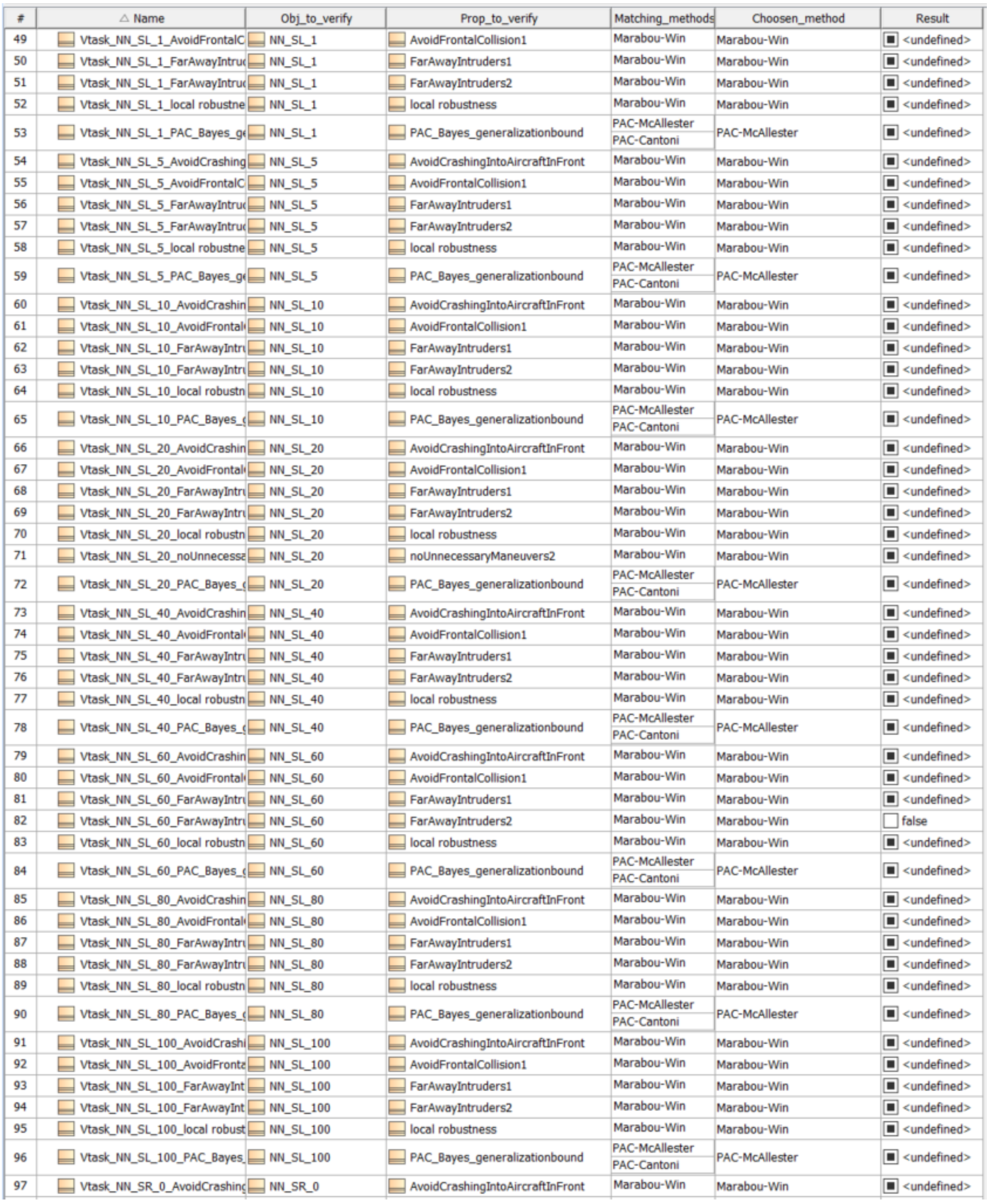
Name	Obj Applied On	Subdomain	Formal_class	Domain	Describtion	Filename
AvoidCrashingIntoAircraftInFront	NN_COC_0 NN_COC_1 NN_COC_10 NN_COC_20 NN_COC_40 NN_COC_5 NN_SL_0 NN_SL_1 NN_SL_10 NN_SL_100 NN_SL_20 NN_SL_40 NN_SL_5 NN_SL_60 NN_SL_80 NN_SR_0 NN_SR_1 NN_SR_10 NN_SR_100 NN_SR_20 NN_SR_40 NN_SR_5 NN_SR_60 NN_SR_80 NN_WL_0 NN_WL_1 NN_WL_10 NN_WL_100 NN_WL_20 NN_WL_40 NN_WL_5 NN_WL_60 NN_WL_80 NN_WR_0 NN_WR_1 NN_WR_10 NN_WR_100 NN_WR_20 NN_WR_40 NN_WR_5 NN_WR_60 NN_WR_80	Application-specific-Safety-Property	Local-Input-Output-Property	Safety	If the intruder is flighting in front of you in the same direction but with a lower speed, do not advice COC (phi4)	acas_property_4.txt

Figure B.1: Example Property of HCAS



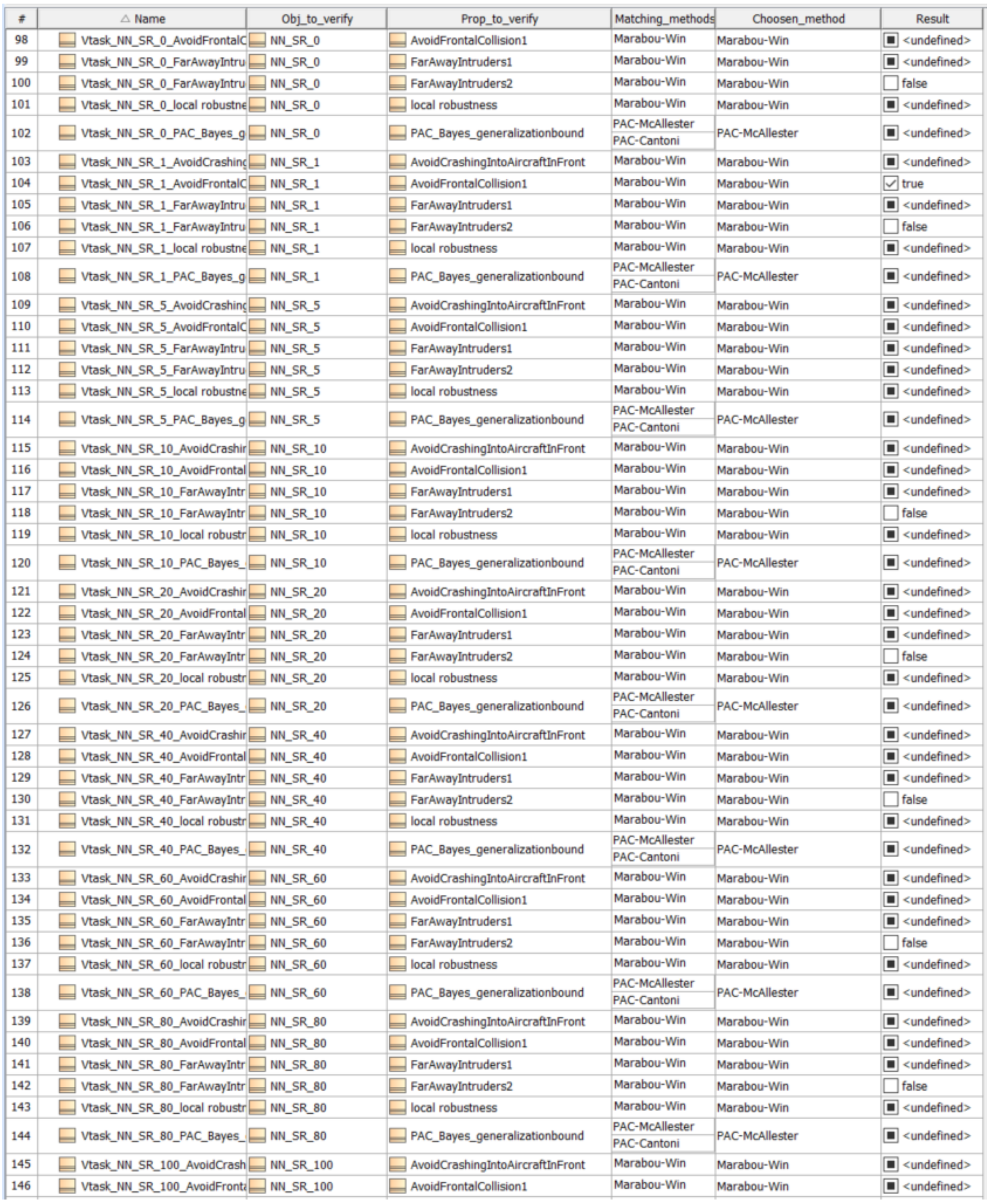
#	△ Name	Obj_to_verify	Prop_to_verify	Matching_methods	Choosen_method	Result
1	Vtask_NN_COC_0_AvoidCrashi	NN_COC_0	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
2	Vtask_NN_COC_0_AvoidFronta	NN_COC_0	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
3	Vtask_NN_COC_0_AvoidIntrude	NN_COC_0	AvoidIntruderFromLeft	Marabou-Win	Marabou-Win	■ <undefined>
4	Vtask_NN_COC_0_FarAwayIntr	NN_COC_0	FarAwayIntruders1	Marabou-Win	Marabou-Win	✓ true
5	Vtask_NN_COC_0_local robustr	NN_COC_0	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
6	Vtask_NN_COC_0_noUnnecess	NN_COC_0	noUnnecessaryManeuvers1	Marabou-Win	Marabou-Win	■ <undefined>
7	Vtask_NN_COC_0_PAC_Bayes_	NN_COC_0	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
8	Vtask_NN_COC_1_AvoidCrashi	NN_COC_1	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
9	Vtask_NN_COC_1_AvoidFronta	NN_COC_1	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
10	Vtask_NN_COC_1_FarAwayIntr	NN_COC_1	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
11	Vtask_NN_COC_1_local robustr	NN_COC_1	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
12	Vtask_NN_COC_1_PAC_Bayes_	NN_COC_1	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
13	Vtask_NN_COC_5_AvoidCrashi	NN_COC_5	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
14	Vtask_NN_COC_5_AvoidFronta	NN_COC_5	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
15	Vtask_NN_COC_5_FarAwayIntr	NN_COC_5	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
16	Vtask_NN_COC_5_local robustr	NN_COC_5	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
17	Vtask_NN_COC_5_PAC_Bayes_	NN_COC_5	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
18	Vtask_NN_COC_10_AvoidCrash	NN_COC_10	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	✓ true
19	Vtask_NN_COC_10_AvoidFront	NN_COC_10	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
20	Vtask_NN_COC_10_FarAwayIn	NN_COC_10	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
21	Vtask_NN_COC_10_local robus	NN_COC_10	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
22	Vtask_NN_COC_10_PAC_Bayes	NN_COC_10	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
23	Vtask_NN_COC_20_AvoidCrash	NN_COC_20	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	✓ true
24	Vtask_NN_COC_20_AvoidFront	NN_COC_20	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	✓ true
25	Vtask_NN_COC_20_FarAwayIn	NN_COC_20	FarAwayIntruders1	Marabou-Win	Marabou-Win	✓ true
26	Vtask_NN_COC_20_local robus	NN_COC_20	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
27	Vtask_NN_COC_20_PAC_Bayes	NN_COC_20	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
28	Vtask_NN_COC_40_AvoidCrash	NN_COC_40	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	✓ true
29	Vtask_NN_COC_40_AvoidFront	NN_COC_40	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
30	Vtask_NN_COC_40_FarAwayIn	NN_COC_40	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
31	Vtask_NN_COC_40_local robus	NN_COC_40	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
32	Vtask_NN_COC_40_PAC_Bayes	NN_COC_40	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
33	Vtask_NN_COC_60_FarAwayIn	NN_COC_60	FarAwayIntruders1	Marabou-Win	Marabou-Win	✓ true
34	Vtask_NN_COC_60_local robus	NN_COC_60	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
35	Vtask_NN_COC_60_PAC_Bayes	NN_COC_60	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
36	Vtask_NN_COC_80_FarAwayIn	NN_COC_80	FarAwayIntruders1	Marabou-Win	Marabou-Win	✓ true
37	Vtask_NN_COC_80_local robus	NN_COC_80	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
38	Vtask_NN_COC_80_PAC_Bayes	NN_COC_80	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
39	Vtask_NN_COC_100_FarAwayI	NN_COC_100	FarAwayIntruders1	Marabou-Win	Marabou-Win	✓ true
40	Vtask_NN_COC_100_local robu	NN_COC_100	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
41	Vtask_NN_COC_100_PAC_Baye	NN_COC_100	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
42	Vtask_NN_SL_0_AvoidCrashing	NN_SL_0	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
43	Vtask_NN_SL_0_AvoidFrontalC	NN_SL_0	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
44	Vtask_NN_SL_0_FarAwayIntru	NN_SL_0	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
45	Vtask_NN_SL_0_FarAwayIntru	NN_SL_0	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
46	Vtask_NN_SL_0_local robustne	NN_SL_0	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
47	Vtask_NN_SL_0_PAC_Bayes_g	NN_SL_0	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
48	Vtask_NN_SL_1_AvoidCrashing	NN_SL_1	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>

Figure B.2: Section of the Verification Task table



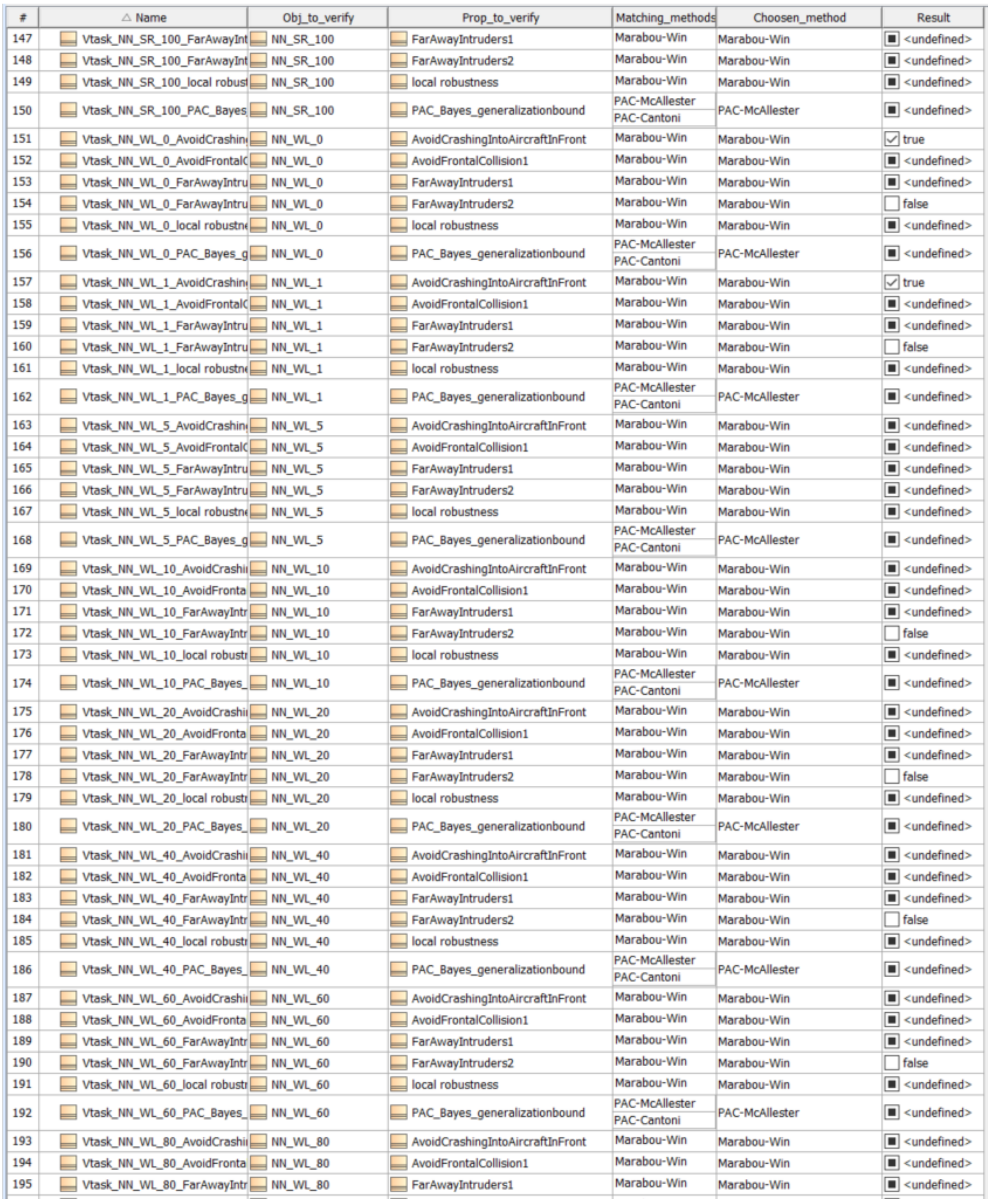
#	△ Name	Obj_to_verify	Prop_to_verify	Matching_methods	Choosen_method	Result
49	Vtask_NN_SL_1_AvoidFrontalC	NN_SL_1	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
50	Vtask_NN_SL_1_FarAwayIntru	NN_SL_1	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
51	Vtask_NN_SL_1_FarAwayIntru	NN_SL_1	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
52	Vtask_NN_SL_1_local robustne	NN_SL_1	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
53	Vtask_NN_SL_1_PAC_Bayes_g	NN_SL_1	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
54	Vtask_NN_SL_5_AvoidCrashing	NN_SL_5	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
55	Vtask_NN_SL_5_AvoidFrontalC	NN_SL_5	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
56	Vtask_NN_SL_5_FarAwayIntru	NN_SL_5	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
57	Vtask_NN_SL_5_FarAwayIntru	NN_SL_5	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
58	Vtask_NN_SL_5_local robustne	NN_SL_5	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
59	Vtask_NN_SL_5_PAC_Bayes_g	NN_SL_5	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
60	Vtask_NN_SL_10_AvoidCrashin	NN_SL_10	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
61	Vtask_NN_SL_10_AvoidFrontal	NN_SL_10	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
62	Vtask_NN_SL_10_FarAwayIntru	NN_SL_10	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
63	Vtask_NN_SL_10_FarAwayIntru	NN_SL_10	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
64	Vtask_NN_SL_10_local robustn	NN_SL_10	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
65	Vtask_NN_SL_10_PAC_Bayes_	NN_SL_10	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
66	Vtask_NN_SL_20_AvoidCrashin	NN_SL_20	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
67	Vtask_NN_SL_20_AvoidFrontal	NN_SL_20	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
68	Vtask_NN_SL_20_FarAwayIntru	NN_SL_20	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
69	Vtask_NN_SL_20_FarAwayIntru	NN_SL_20	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
70	Vtask_NN_SL_20_local robustn	NN_SL_20	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
71	Vtask_NN_SL_20_noUnnecessa	NN_SL_20	noUnnecessaryManeuvers2	Marabou-Win	Marabou-Win	■ <undefined>
72	Vtask_NN_SL_20_PAC_Bayes_	NN_SL_20	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
73	Vtask_NN_SL_40_AvoidCrashin	NN_SL_40	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
74	Vtask_NN_SL_40_AvoidFrontal	NN_SL_40	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
75	Vtask_NN_SL_40_FarAwayIntru	NN_SL_40	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
76	Vtask_NN_SL_40_FarAwayIntru	NN_SL_40	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
77	Vtask_NN_SL_40_local robustn	NN_SL_40	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
78	Vtask_NN_SL_40_PAC_Bayes_	NN_SL_40	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
79	Vtask_NN_SL_60_AvoidCrashin	NN_SL_60	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
80	Vtask_NN_SL_60_AvoidFrontal	NN_SL_60	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
81	Vtask_NN_SL_60_FarAwayIntru	NN_SL_60	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
82	Vtask_NN_SL_60_FarAwayIntru	NN_SL_60	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
83	Vtask_NN_SL_60_local robustn	NN_SL_60	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
84	Vtask_NN_SL_60_PAC_Bayes_	NN_SL_60	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
85	Vtask_NN_SL_80_AvoidCrashin	NN_SL_80	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
86	Vtask_NN_SL_80_AvoidFrontal	NN_SL_80	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
87	Vtask_NN_SL_80_FarAwayIntru	NN_SL_80	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
88	Vtask_NN_SL_80_FarAwayIntru	NN_SL_80	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
89	Vtask_NN_SL_80_local robustn	NN_SL_80	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
90	Vtask_NN_SL_80_PAC_Bayes_	NN_SL_80	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
91	Vtask_NN_SL_100_AvoidCrashi	NN_SL_100	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
92	Vtask_NN_SL_100_AvoidFronta	NN_SL_100	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
93	Vtask_NN_SL_100_FarAwayInt	NN_SL_100	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
94	Vtask_NN_SL_100_FarAwayInt	NN_SL_100	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
95	Vtask_NN_SL_100_local robust	NN_SL_100	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
96	Vtask_NN_SL_100_PAC_Bayes_	NN_SL_100	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
97	Vtask_NN_SR_0_AvoidCrashing	NN_SR_0	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>

Figure B.3: Section of the Verification Task table



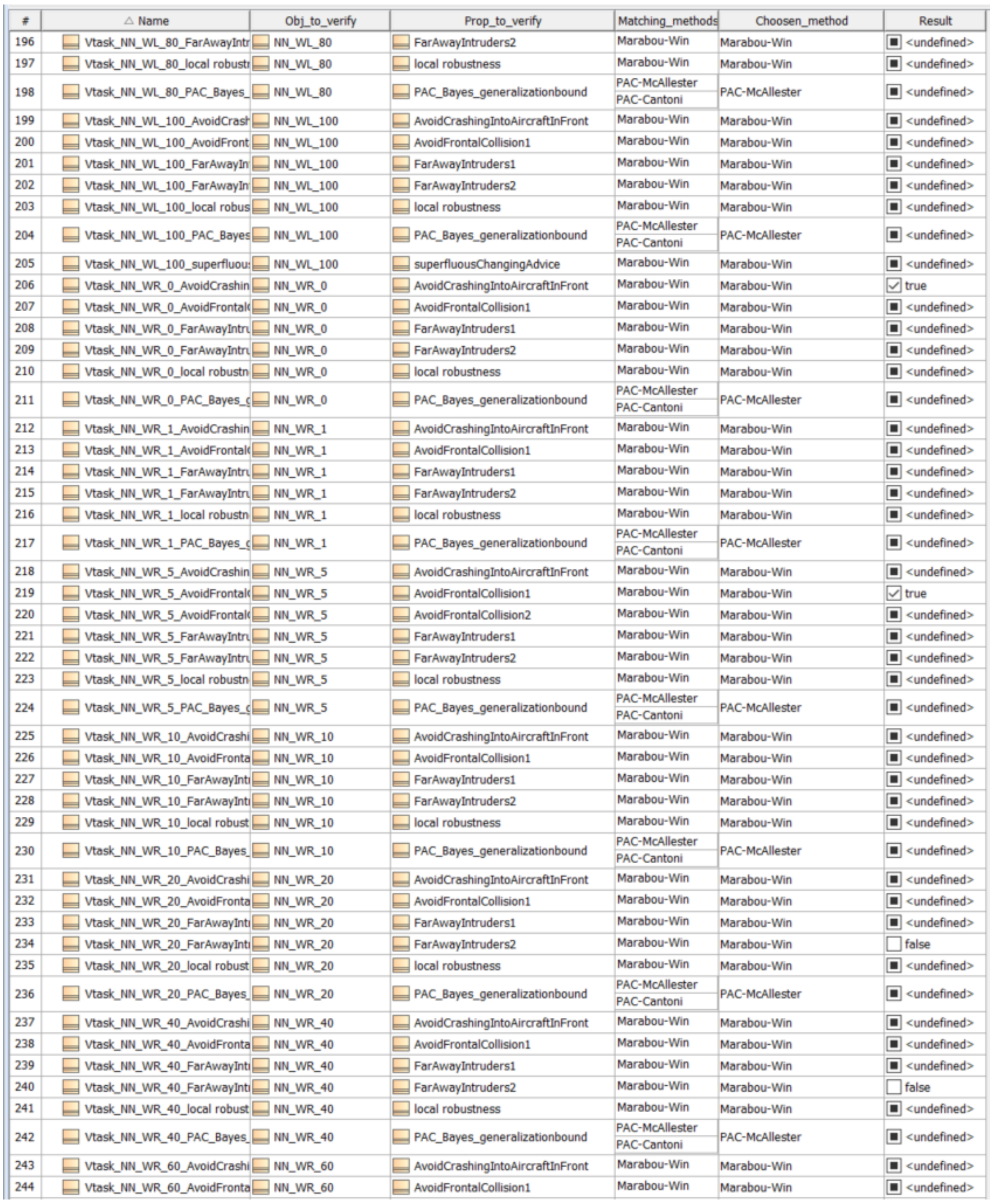
#	△ Name	Obj_to_verify	Prop_to_verify	Matching_methods	Choosen_method	Result
98	Vtask_NN_SR_0_AvoidFrontalC	NN_SR_0	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
99	Vtask_NN_SR_0_FarAwayIntru	NN_SR_0	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
100	Vtask_NN_SR_0_FarAwayIntru	NN_SR_0	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
101	Vtask_NN_SR_0_local robustne	NN_SR_0	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
102	Vtask_NN_SR_0_PAC_Bayes_g	NN_SR_0	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
103	Vtask_NN_SR_1_AvoidCrashinc	NN_SR_1	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
104	Vtask_NN_SR_1_AvoidFrontalC	NN_SR_1	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	✓ true
105	Vtask_NN_SR_1_FarAwayIntru	NN_SR_1	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
106	Vtask_NN_SR_1_FarAwayIntru	NN_SR_1	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
107	Vtask_NN_SR_1_local robustne	NN_SR_1	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
108	Vtask_NN_SR_1_PAC_Bayes_g	NN_SR_1	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
109	Vtask_NN_SR_5_AvoidCrashinc	NN_SR_5	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
110	Vtask_NN_SR_5_AvoidFrontalC	NN_SR_5	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
111	Vtask_NN_SR_5_FarAwayIntru	NN_SR_5	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
112	Vtask_NN_SR_5_FarAwayIntru	NN_SR_5	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
113	Vtask_NN_SR_5_local robustne	NN_SR_5	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
114	Vtask_NN_SR_5_PAC_Bayes_g	NN_SR_5	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
115	Vtask_NN_SR_10_AvoidCrashir	NN_SR_10	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
116	Vtask_NN_SR_10_AvoidFrontal	NN_SR_10	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
117	Vtask_NN_SR_10_FarAwayIntr	NN_SR_10	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
118	Vtask_NN_SR_10_FarAwayIntr	NN_SR_10	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
119	Vtask_NN_SR_10_local robustr	NN_SR_10	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
120	Vtask_NN_SR_10_PAC_Bayes_	NN_SR_10	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
121	Vtask_NN_SR_20_AvoidCrashir	NN_SR_20	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
122	Vtask_NN_SR_20_AvoidFrontal	NN_SR_20	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
123	Vtask_NN_SR_20_FarAwayIntr	NN_SR_20	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
124	Vtask_NN_SR_20_FarAwayIntr	NN_SR_20	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
125	Vtask_NN_SR_20_local robustr	NN_SR_20	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
126	Vtask_NN_SR_20_PAC_Bayes_	NN_SR_20	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
127	Vtask_NN_SR_40_AvoidCrashir	NN_SR_40	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
128	Vtask_NN_SR_40_AvoidFrontal	NN_SR_40	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
129	Vtask_NN_SR_40_FarAwayIntr	NN_SR_40	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
130	Vtask_NN_SR_40_FarAwayIntr	NN_SR_40	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
131	Vtask_NN_SR_40_local robustr	NN_SR_40	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
132	Vtask_NN_SR_40_PAC_Bayes_	NN_SR_40	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
133	Vtask_NN_SR_60_AvoidCrashir	NN_SR_60	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
134	Vtask_NN_SR_60_AvoidFrontal	NN_SR_60	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
135	Vtask_NN_SR_60_FarAwayIntr	NN_SR_60	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
136	Vtask_NN_SR_60_FarAwayIntr	NN_SR_60	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
137	Vtask_NN_SR_60_local robustr	NN_SR_60	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
138	Vtask_NN_SR_60_PAC_Bayes_	NN_SR_60	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
139	Vtask_NN_SR_80_AvoidCrashir	NN_SR_80	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
140	Vtask_NN_SR_80_AvoidFrontal	NN_SR_80	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
141	Vtask_NN_SR_80_FarAwayIntr	NN_SR_80	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
142	Vtask_NN_SR_80_FarAwayIntr	NN_SR_80	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
143	Vtask_NN_SR_80_local robustr	NN_SR_80	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
144	Vtask_NN_SR_80_PAC_Bayes_	NN_SR_80	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
145	Vtask_NN_SR_100_AvoidCrash	NN_SR_100	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
146	Vtask_NN_SR_100_AvoidFront	NN_SR_100	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>

Figure B.4: Section of the Verification Task table



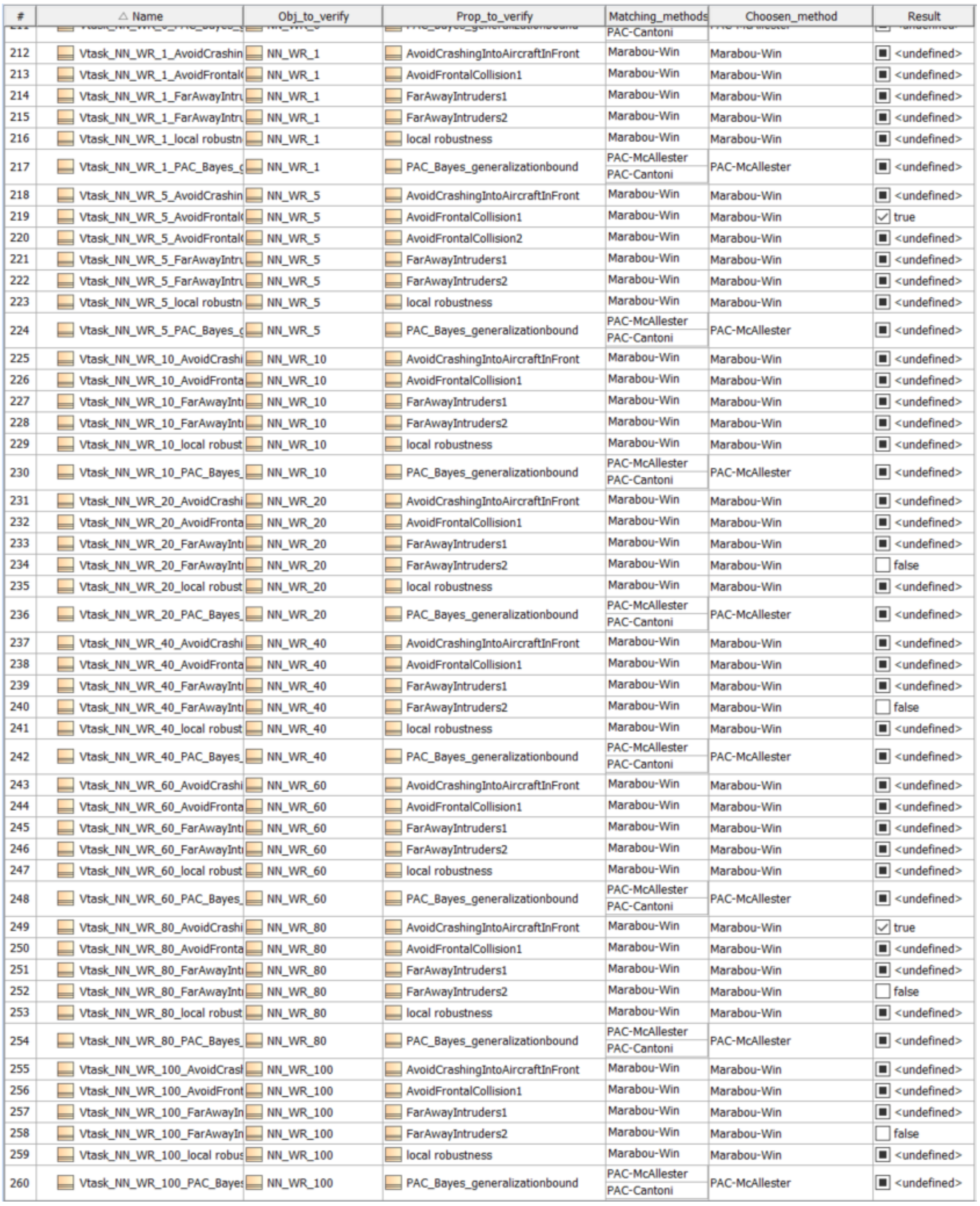
#	△ Name	Obj_to_verify	Prop_to_verify	Matching_methods	Choosen_method	Result
147	Vtask_NN_SR_100_FarAwayInt	NN_SR_100	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
148	Vtask_NN_SR_100_FarAwayInt	NN_SR_100	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
149	Vtask_NN_SR_100_local robust	NN_SR_100	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
150	Vtask_NN_SR_100_PAC_Bayes	NN_SR_100	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
151	Vtask_NN_WL_0_AvoidCrashin	NN_WL_0	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	✓ true
152	Vtask_NN_WL_0_AvoidFrontalC	NN_WL_0	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
153	Vtask_NN_WL_0_FarAwayIntru	NN_WL_0	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
154	Vtask_NN_WL_0_FarAwayIntru	NN_WL_0	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
155	Vtask_NN_WL_0_local robustne	NN_WL_0	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
156	Vtask_NN_WL_0_PAC_Bayes_g	NN_WL_0	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
157	Vtask_NN_WL_1_AvoidCrashin	NN_WL_1	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	✓ true
158	Vtask_NN_WL_1_AvoidFrontalC	NN_WL_1	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
159	Vtask_NN_WL_1_FarAwayIntru	NN_WL_1	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
160	Vtask_NN_WL_1_FarAwayIntru	NN_WL_1	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
161	Vtask_NN_WL_1_local robustne	NN_WL_1	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
162	Vtask_NN_WL_1_PAC_Bayes_g	NN_WL_1	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
163	Vtask_NN_WL_5_AvoidCrashin	NN_WL_5	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
164	Vtask_NN_WL_5_AvoidFrontalC	NN_WL_5	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
165	Vtask_NN_WL_5_FarAwayIntru	NN_WL_5	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
166	Vtask_NN_WL_5_FarAwayIntru	NN_WL_5	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
167	Vtask_NN_WL_5_local robustne	NN_WL_5	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
168	Vtask_NN_WL_5_PAC_Bayes_g	NN_WL_5	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
169	Vtask_NN_WL_10_AvoidCrashi	NN_WL_10	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
170	Vtask_NN_WL_10_AvoidFronta	NN_WL_10	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
171	Vtask_NN_WL_10_FarAwayIntr	NN_WL_10	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
172	Vtask_NN_WL_10_FarAwayIntr	NN_WL_10	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
173	Vtask_NN_WL_10_local robust	NN_WL_10	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
174	Vtask_NN_WL_10_PAC_Bayes_	NN_WL_10	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
175	Vtask_NN_WL_20_AvoidCrashi	NN_WL_20	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
176	Vtask_NN_WL_20_AvoidFronta	NN_WL_20	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
177	Vtask_NN_WL_20_FarAwayIntr	NN_WL_20	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
178	Vtask_NN_WL_20_FarAwayIntr	NN_WL_20	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
179	Vtask_NN_WL_20_local robust	NN_WL_20	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
180	Vtask_NN_WL_20_PAC_Bayes_	NN_WL_20	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
181	Vtask_NN_WL_40_AvoidCrashi	NN_WL_40	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
182	Vtask_NN_WL_40_AvoidFronta	NN_WL_40	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
183	Vtask_NN_WL_40_FarAwayIntr	NN_WL_40	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
184	Vtask_NN_WL_40_FarAwayIntr	NN_WL_40	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
185	Vtask_NN_WL_40_local robust	NN_WL_40	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
186	Vtask_NN_WL_40_PAC_Bayes_	NN_WL_40	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
187	Vtask_NN_WL_60_AvoidCrashi	NN_WL_60	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
188	Vtask_NN_WL_60_AvoidFronta	NN_WL_60	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
189	Vtask_NN_WL_60_FarAwayIntr	NN_WL_60	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
190	Vtask_NN_WL_60_FarAwayIntr	NN_WL_60	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
191	Vtask_NN_WL_60_local robust	NN_WL_60	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
192	Vtask_NN_WL_60_PAC_Bayes_	NN_WL_60	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
193	Vtask_NN_WL_80_AvoidCrashi	NN_WL_80	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
194	Vtask_NN_WL_80_AvoidFronta	NN_WL_80	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
195	Vtask_NN_WL_80_FarAwayIntr	NN_WL_80	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>

Figure B.5: Section of the Verification Task table



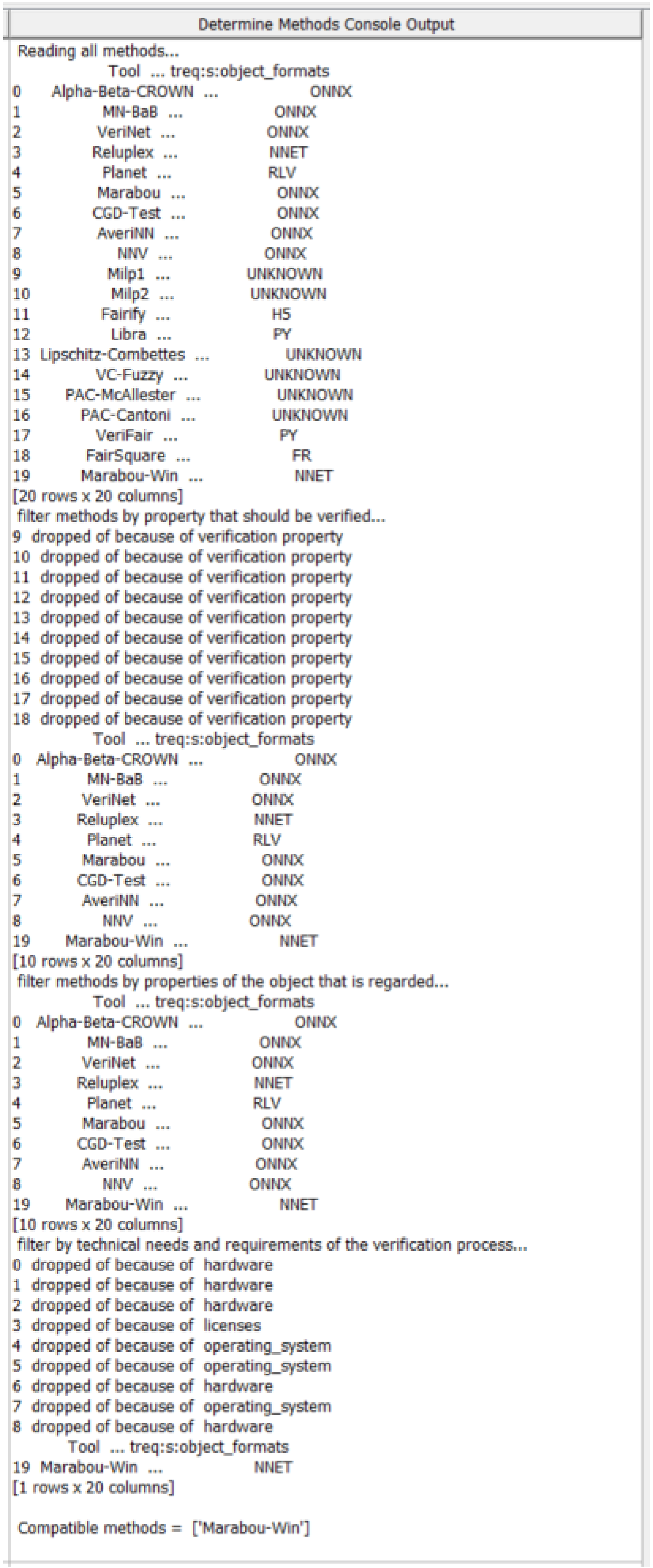
#	Name	Obj_to_verify	Prop_to_verify	Matching_methods	Choosen_method	Result
196	Vtask_NN_WL_80_FarAwayIntr	NN_WL_80	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
197	Vtask_NN_WL_80_local robustr	NN_WL_80	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
198	Vtask_NN_WL_80_PAC_Bayes_	NN_WL_80	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
199	Vtask_NN_WL_100_AvoidCrash	NN_WL_100	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
200	Vtask_NN_WL_100_AvoidFront	NN_WL_100	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
201	Vtask_NN_WL_100_FarAwayIn	NN_WL_100	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
202	Vtask_NN_WL_100_FarAwayIn	NN_WL_100	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
203	Vtask_NN_WL_100_local robus	NN_WL_100	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
204	Vtask_NN_WL_100_PAC_Bayes	NN_WL_100	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
205	Vtask_NN_WL_100_superfluou	NN_WL_100	superfluousChangingAdvice	Marabou-Win	Marabou-Win	■ <undefined>
206	Vtask_NN_WR_0_AvoidCrashin	NN_WR_0	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	✓ true
207	Vtask_NN_WR_0_AvoidFrontal	NN_WR_0	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
208	Vtask_NN_WR_0_FarAwayIntru	NN_WR_0	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
209	Vtask_NN_WR_0_FarAwayIntru	NN_WR_0	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
210	Vtask_NN_WR_0_local robustn	NN_WR_0	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
211	Vtask_NN_WR_0_PAC_Bayes_	NN_WR_0	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
212	Vtask_NN_WR_1_AvoidCrashin	NN_WR_1	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
213	Vtask_NN_WR_1_AvoidFrontal	NN_WR_1	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
214	Vtask_NN_WR_1_FarAwayIntru	NN_WR_1	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
215	Vtask_NN_WR_1_FarAwayIntru	NN_WR_1	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
216	Vtask_NN_WR_1_local robustn	NN_WR_1	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
217	Vtask_NN_WR_1_PAC_Bayes_	NN_WR_1	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
218	Vtask_NN_WR_5_AvoidCrashin	NN_WR_5	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
219	Vtask_NN_WR_5_AvoidFrontal	NN_WR_5	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	✓ true
220	Vtask_NN_WR_5_AvoidFrontal	NN_WR_5	AvoidFrontalCollision2	Marabou-Win	Marabou-Win	■ <undefined>
221	Vtask_NN_WR_5_FarAwayIntru	NN_WR_5	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
222	Vtask_NN_WR_5_FarAwayIntru	NN_WR_5	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
223	Vtask_NN_WR_5_local robustn	NN_WR_5	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
224	Vtask_NN_WR_5_PAC_Bayes_	NN_WR_5	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
225	Vtask_NN_WR_10_AvoidCrashi	NN_WR_10	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
226	Vtask_NN_WR_10_AvoidFronta	NN_WR_10	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
227	Vtask_NN_WR_10_FarAwayInt	NN_WR_10	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
228	Vtask_NN_WR_10_FarAwayInt	NN_WR_10	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
229	Vtask_NN_WR_10_local robust	NN_WR_10	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
230	Vtask_NN_WR_10_PAC_Bayes_	NN_WR_10	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
231	Vtask_NN_WR_20_AvoidCrashi	NN_WR_20	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
232	Vtask_NN_WR_20_AvoidFronta	NN_WR_20	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
233	Vtask_NN_WR_20_FarAwayInt	NN_WR_20	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
234	Vtask_NN_WR_20_FarAwayInt	NN_WR_20	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
235	Vtask_NN_WR_20_local robust	NN_WR_20	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
236	Vtask_NN_WR_20_PAC_Bayes_	NN_WR_20	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
237	Vtask_NN_WR_40_AvoidCrashi	NN_WR_40	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
238	Vtask_NN_WR_40_AvoidFronta	NN_WR_40	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
239	Vtask_NN_WR_40_FarAwayInt	NN_WR_40	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
240	Vtask_NN_WR_40_FarAwayInt	NN_WR_40	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
241	Vtask_NN_WR_40_local robust	NN_WR_40	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
242	Vtask_NN_WR_40_PAC_Bayes_	NN_WR_40	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
243	Vtask_NN_WR_60_AvoidCrashi	NN_WR_60	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
244	Vtask_NN_WR_60_AvoidFronta	NN_WR_60	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>

Figure B.6: Section of the Verification Task table



#	△ Name	Obj_to_verify	Prop_to_verify	Matching_methods	Choosen_method	Result
				PAC-Cantoni		
212	Vtask_NN_WR_1_AvoidCrashin	NN_WR_1	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
213	Vtask_NN_WR_1_AvoidFrontal(	NN_WR_1	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
214	Vtask_NN_WR_1_FarAwayIntru	NN_WR_1	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
215	Vtask_NN_WR_1_FarAwayIntru	NN_WR_1	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
216	Vtask_NN_WR_1_local robustn	NN_WR_1	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
217	Vtask_NN_WR_1_PAC_Bayes_(	NN_WR_1	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
218	Vtask_NN_WR_5_AvoidCrashin	NN_WR_5	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
219	Vtask_NN_WR_5_AvoidFrontal(	NN_WR_5	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	✓ true
220	Vtask_NN_WR_5_AvoidFrontal(	NN_WR_5	AvoidFrontalCollision2	Marabou-Win	Marabou-Win	■ <undefined>
221	Vtask_NN_WR_5_FarAwayIntru	NN_WR_5	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
222	Vtask_NN_WR_5_FarAwayIntru	NN_WR_5	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
223	Vtask_NN_WR_5_local robustn	NN_WR_5	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
224	Vtask_NN_WR_5_PAC_Bayes_(	NN_WR_5	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
225	Vtask_NN_WR_10_AvoidCrashi	NN_WR_10	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
226	Vtask_NN_WR_10_AvoidFronta	NN_WR_10	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
227	Vtask_NN_WR_10_FarAwayInt	NN_WR_10	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
228	Vtask_NN_WR_10_FarAwayInt	NN_WR_10	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
229	Vtask_NN_WR_10_local robust	NN_WR_10	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
230	Vtask_NN_WR_10_PAC_Bayes_	NN_WR_10	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
231	Vtask_NN_WR_20_AvoidCrashi	NN_WR_20	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
232	Vtask_NN_WR_20_AvoidFronta	NN_WR_20	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
233	Vtask_NN_WR_20_FarAwayInt	NN_WR_20	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
234	Vtask_NN_WR_20_FarAwayInt	NN_WR_20	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
235	Vtask_NN_WR_20_local robust	NN_WR_20	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
236	Vtask_NN_WR_20_PAC_Bayes_	NN_WR_20	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
237	Vtask_NN_WR_40_AvoidCrashi	NN_WR_40	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
238	Vtask_NN_WR_40_AvoidFronta	NN_WR_40	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
239	Vtask_NN_WR_40_FarAwayInt	NN_WR_40	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
240	Vtask_NN_WR_40_FarAwayInt	NN_WR_40	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
241	Vtask_NN_WR_40_local robust	NN_WR_40	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
242	Vtask_NN_WR_40_PAC_Bayes_	NN_WR_40	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
243	Vtask_NN_WR_60_AvoidCrashi	NN_WR_60	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
244	Vtask_NN_WR_60_AvoidFronta	NN_WR_60	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
245	Vtask_NN_WR_60_FarAwayInt	NN_WR_60	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
246	Vtask_NN_WR_60_FarAwayInt	NN_WR_60	FarAwayIntruders2	Marabou-Win	Marabou-Win	■ <undefined>
247	Vtask_NN_WR_60_local robust	NN_WR_60	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
248	Vtask_NN_WR_60_PAC_Bayes_	NN_WR_60	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
249	Vtask_NN_WR_80_AvoidCrashi	NN_WR_80	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	✓ true
250	Vtask_NN_WR_80_AvoidFronta	NN_WR_80	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
251	Vtask_NN_WR_80_FarAwayInt	NN_WR_80	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
252	Vtask_NN_WR_80_FarAwayInt	NN_WR_80	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
253	Vtask_NN_WR_80_local robust	NN_WR_80	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
254	Vtask_NN_WR_80_PAC_Bayes_	NN_WR_80	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>
255	Vtask_NN_WR_100_AvoidCrasl	NN_WR_100	AvoidCrashingIntoAircraftInFront	Marabou-Win	Marabou-Win	■ <undefined>
256	Vtask_NN_WR_100_AvoidFront	NN_WR_100	AvoidFrontalCollision1	Marabou-Win	Marabou-Win	■ <undefined>
257	Vtask_NN_WR_100_FarAwayIn	NN_WR_100	FarAwayIntruders1	Marabou-Win	Marabou-Win	■ <undefined>
258	Vtask_NN_WR_100_FarAwayIn	NN_WR_100	FarAwayIntruders2	Marabou-Win	Marabou-Win	☐ false
259	Vtask_NN_WR_100_local robus	NN_WR_100	local robustness	Marabou-Win	Marabou-Win	■ <undefined>
260	Vtask_NN_WR_100_PAC_Baye(	NN_WR_100	PAC_Bayes_generalizationbound	PAC-McAllester PAC-Cantoni	PAC-McAllester	■ <undefined>

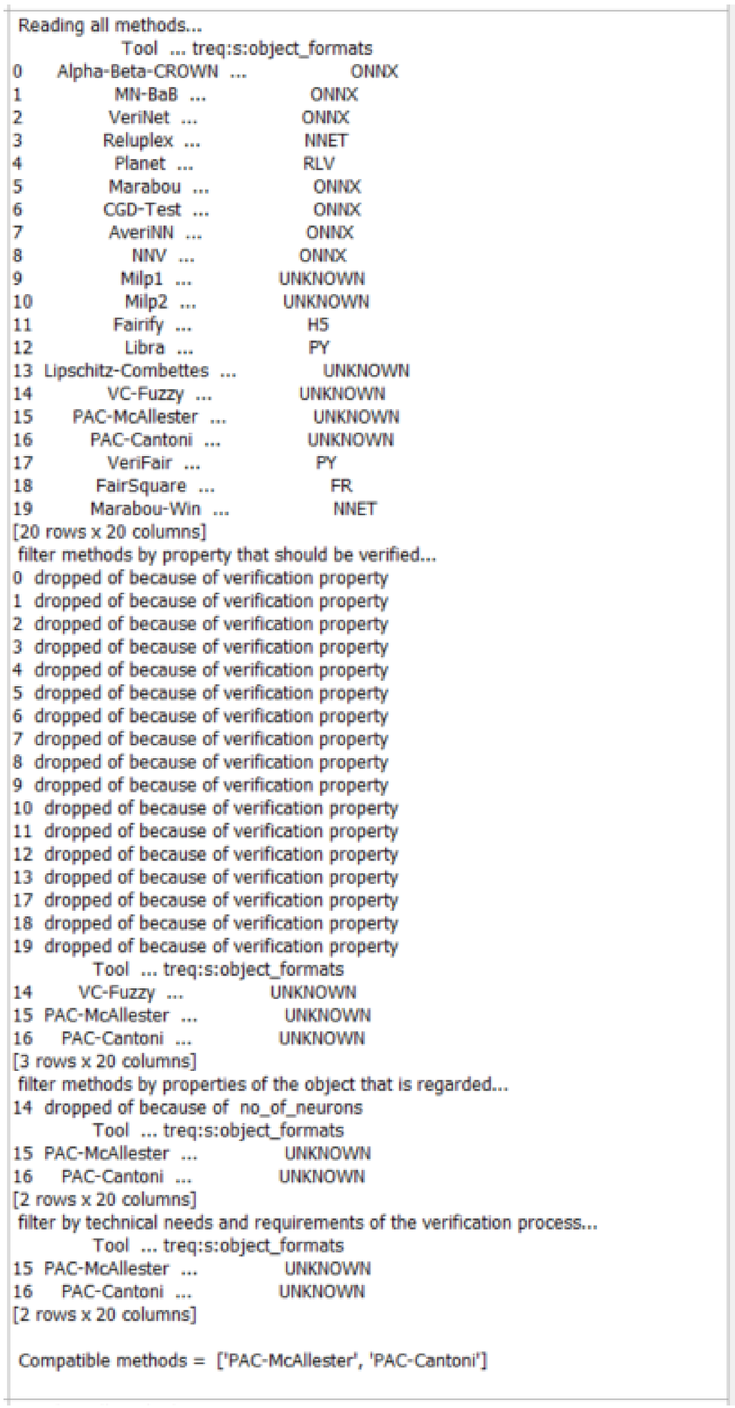
Figure B.7: Section of the Verification Task table



```
Determine Methods Console Output
Reading all methods...
              Tool  ... treq:s:object_formats
0    Alpha-Beta-CROWN  ...                ONNX
1             MN-BaB  ...             ONNX
2             VeriNet  ...            ONNX
3            Reluplex  ...            NNET
4              Planet  ...            RLV
5            Marabou  ...             ONNX
6           CGD-Test  ...             ONNX
7            AveriNN  ...             ONNX
8                NNV  ...             ONNX
9               Milp1  ...          UNKNOWN
10              Milp2  ...          UNKNOWN
11             Fairify  ...            H5
12               Libra  ...            PY
13  Lipschitz-Combettes  ...          UNKNOWN
14           VC-Fuzzy  ...           UNKNOWN
15      PAC-McAllester  ...          UNKNOWN
16        PAC-Cantoni  ...           UNKNOWN
17           VeriFair  ...             PY
18         FairSquare  ...             FR
19        Marabou-Win  ...             NNET
[20 rows x 20 columns]
 filter methods by property that should be verified...
9  dropped of because of verification property
10  dropped of because of verification property
11  dropped of because of verification property
12  dropped of because of verification property
13  dropped of because of verification property
14  dropped of because of verification property
15  dropped of because of verification property
16  dropped of because of verification property
17  dropped of because of verification property
18  dropped of because of verification property
              Tool  ... treq:s:object_formats
0  Alpha-Beta-CROWN  ...                ONNX
1            MN-BaB  ...             ONNX
2           VeriNet  ...            ONNX
3          Reluplex  ...            NNET
4            Planet  ...            RLV
5           Marabou  ...             ONNX
6          CGD-Test  ...             ONNX
7           AveriNN  ...            ONNX
8               NNV  ...            ONNX
19      Marabou-Win  ...             NNET
[10 rows x 20 columns]
 filter methods by properties of the object that is regarded...
              Tool  ... treq:s:object_formats
0  Alpha-Beta-CROWN  ...                ONNX
1            MN-BaB  ...             ONNX
2           VeriNet  ...            ONNX
3          Reluplex  ...            NNET
4            Planet  ...            RLV
5           Marabou  ...             ONNX
6          CGD-Test  ...             ONNX
7           AveriNN  ...            ONNX
8               NNV  ...            ONNX
19      Marabou-Win  ...             NNET
[10 rows x 20 columns]
 filter by technical needs and requirements of the verification process...
0  dropped of because of  hardware
1  dropped of because of  hardware
2  dropped of because of  hardware
3  dropped of because of  licenses
4  dropped of because of  operating_system
5  dropped of because of  operating_system
6  dropped of because of  hardware
7  dropped of because of  operating_system
8  dropped of because of  hardware
            Tool  ... treq:s:object_formats
19  Marabou-Win  ...             NNET
[1 rows x 20 columns]

 Compatible methods =  ['Marabou-Win']
```

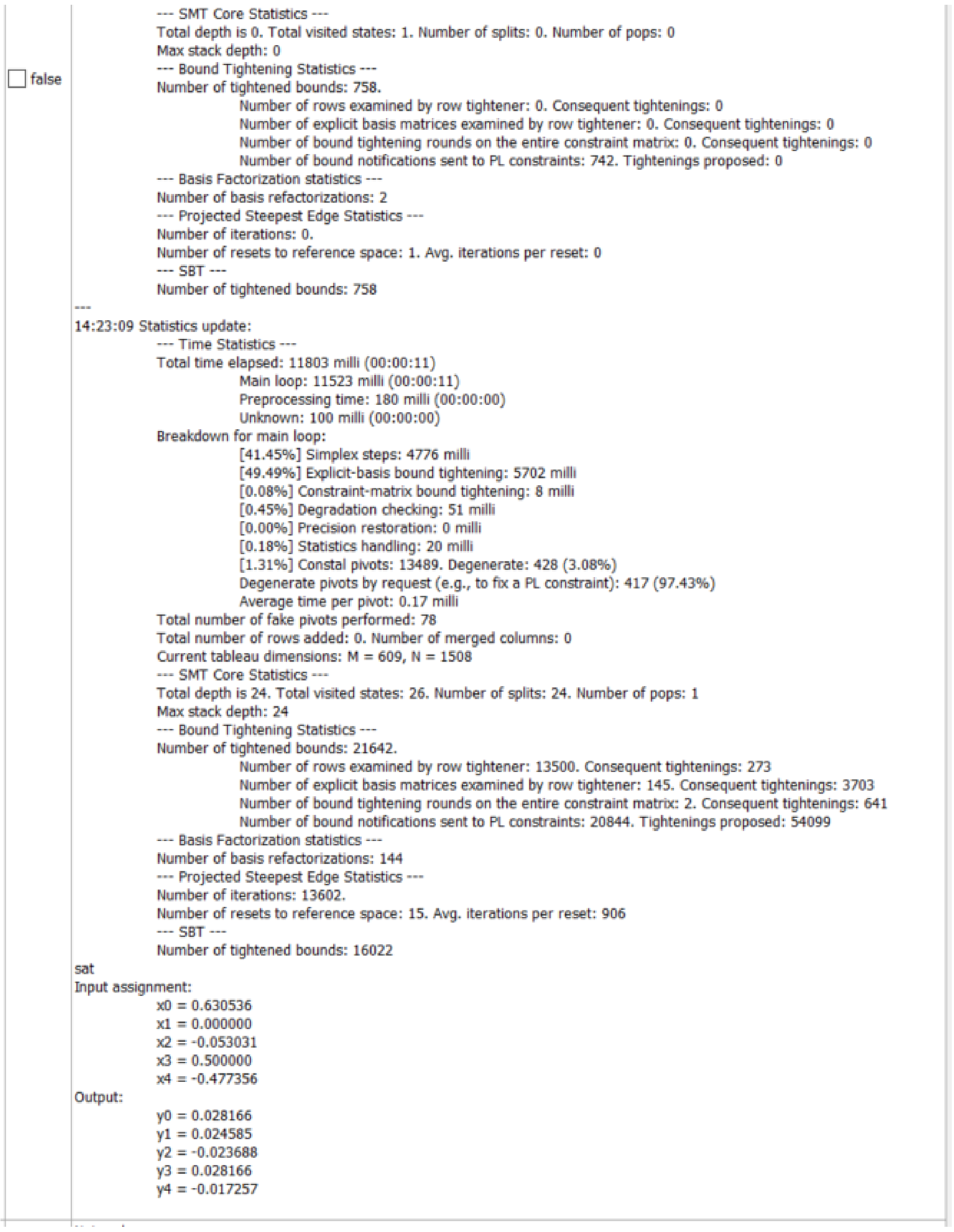
Figure B.8: Example Console Output of *DetermineMethod()*



```
Reading all methods...
                 Tool  ... treq:s:object_formats
0     Alpha-Beta-CROWN  ...                  ONNX
1             MN-BaB  ...                  ONNX
2            VeriNet  ...                  ONNX
3           Reluplex  ...                  NNET
4             Planet  ...                   RLV
5            Marabou  ...                  ONNX
6           CGD-Test  ...                  ONNX
7            AveriNN  ...                  ONNX
8                NNV  ...                  ONNX
9              Milp1  ...               UNKNOWN
10             Milp2  ...               UNKNOWN
11           Fairify  ...                    H5
12             Libra  ...                    PY
13 Lipschitz-Combettes  ...              UNKNOWN
14          VC-Fuzzy  ...               UNKNOWN
15     PAC-McAllester  ...              UNKNOWN
16       PAC-Cantoni  ...               UNKNOWN
17          VeriFair  ...                    PY
18        FairSquare  ...                    FR
19       Marabou-Win  ...                  NNET
[20 rows x 20 columns]
filter methods by property that should be verified...
0  dropped of because of verification property
1  dropped of because of verification property
2  dropped of because of verification property
3  dropped of because of verification property
4  dropped of because of verification property
5  dropped of because of verification property
6  dropped of because of verification property
7  dropped of because of verification property
8  dropped of because of verification property
9  dropped of because of verification property
10  dropped of because of verification property
11  dropped of because of verification property
12  dropped of because of verification property
13  dropped of because of verification property
17  dropped of because of verification property
18  dropped of because of verification property
19  dropped of because of verification property
              Tool  ... treq:s:object_formats
14       VC-Fuzzy  ...               UNKNOWN
15  PAC-McAllester  ...              UNKNOWN
16     PAC-Cantoni  ...              UNKNOWN
[3 rows x 20 columns]
filter methods by properties of the object that is regarded...
14  dropped of because of  no_of_neurons
              Tool  ... treq:s:object_formats
15  PAC-McAllester  ...              UNKNOWN
16     PAC-Cantoni  ...              UNKNOWN
[2 rows x 20 columns]
filter by technical needs and requirements of the verification process...
              Tool  ... treq:s:object_formats
15  PAC-McAllester  ...              UNKNOWN
16     PAC-Cantoni  ...              UNKNOWN
[2 rows x 20 columns]

Compatible methods =  ['PAC-McAllester', 'PAC-Cantoni']
```

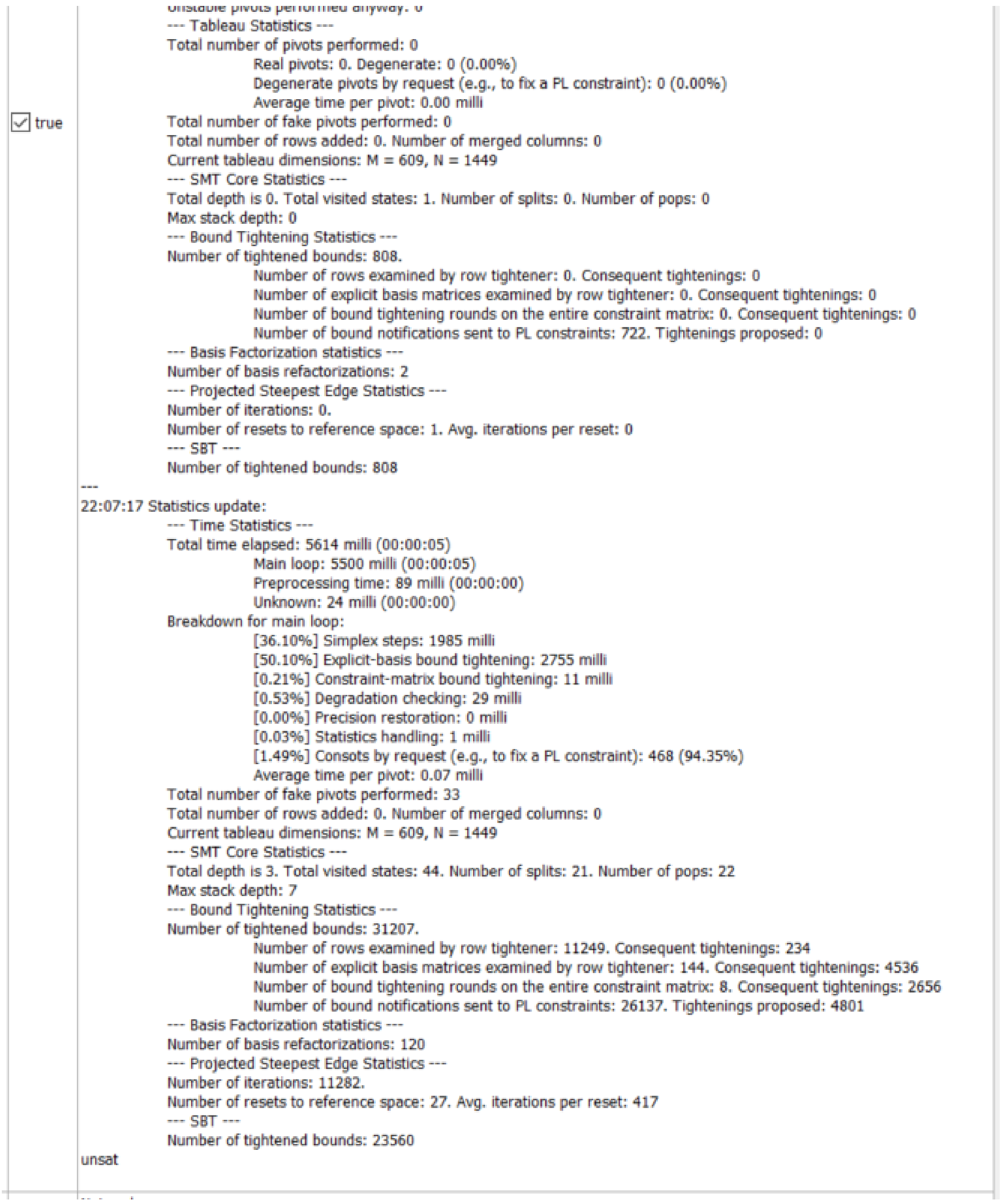
Figure B.9: Example Console Output of *DetermineMethod()*



☐ false

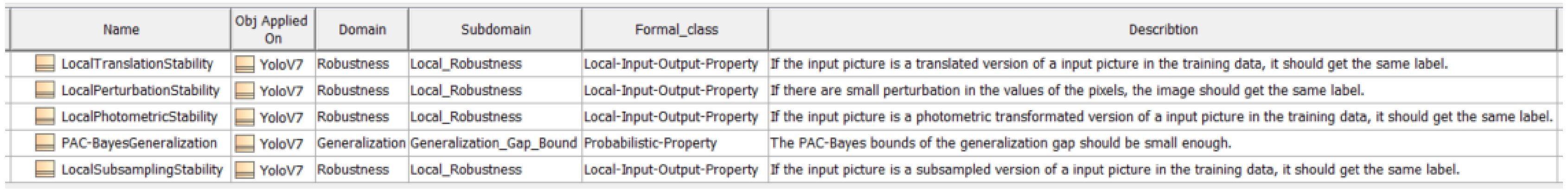
```
        --- SMT Core Statistics ---
        Total depth is 0. Total visited states: 1. Number of splits: 0. Number of pops: 0
        Max stack depth: 0
        --- Bound Tightening Statistics ---
        Number of tightened bounds: 758.
                Number of rows examined by row tightener: 0. Consequent tightenings: 0
                Number of explicit basis matrices examined by row tightener: 0. Consequent tightenings: 0
                Number of bound tightening rounds on the entire constraint matrix: 0. Consequent tightenings: 0
                Number of bound notifications sent to PL constraints: 742. Tightenings proposed: 0
        --- Basis Factorization statistics ---
        Number of basis refactorizations: 2
        --- Projected Steepest Edge Statistics ---
        Number of iterations: 0.
        Number of resets to reference space: 1. Avg. iterations per reset: 0
        --- SBT ---
        Number of tightened bounds: 758
---
14:23:09 Statistics update:
        --- Time Statistics ---
        Total time elapsed: 11803 milli (00:00:11)
                Main loop: 11523 milli (00:00:11)
                Preprocessing time: 180 milli (00:00:00)
                Unknown: 100 milli (00:00:00)
        Breakdown for main loop:
                [41.45%] Simplex steps: 4776 milli
                [49.49%] Explicit-basis bound tightening: 5702 milli
                [0.08%] Constraint-matrix bound tightening: 8 milli
                [0.45%] Degradation checking: 51 milli
                [0.00%] Precision restoration: 0 milli
                [0.18%] Statistics handling: 20 milli
                [1.31%] Constal pivots: 13489. Degenerate: 428 (3.08%)
                Degenerate pivots by request (e.g., to fix a PL constraint): 417 (97.43%)
                Average time per pivot: 0.17 milli
        Total number of fake pivots performed: 78
        Total number of rows added: 0. Number of merged columns: 0
        Current tableau dimensions: M = 609, N = 1508
        --- SMT Core Statistics ---
        Total depth is 24. Total visited states: 26. Number of splits: 24. Number of pops: 1
        Max stack depth: 24
        --- Bound Tightening Statistics ---
        Number of tightened bounds: 21642.
                Number of rows examined by row tightener: 13500. Consequent tightenings: 273
                Number of explicit basis matrices examined by row tightener: 145. Consequent tightenings: 3703
                Number of bound tightening rounds on the entire constraint matrix: 2. Consequent tightenings: 641
                Number of bound notifications sent to PL constraints: 20844. Tightenings proposed: 54099
        --- Basis Factorization statistics ---
        Number of basis refactorizations: 144
        --- Projected Steepest Edge Statistics ---
        Number of iterations: 13602.
        Number of resets to reference space: 15. Avg. iterations per reset: 906
        --- SBT ---
        Number of tightened bounds: 16022
sat
Input assignment:
        x0 = 0.630536
        x1 = 0.000000
        x2 = -0.053031
        x3 = 0.500000
        x4 = -0.477356
Output:
        y0 = 0.028166
        y1 = 0.024585
        y2 = -0.023688
        y3 = 0.028166
        y4 = -0.017257
```

Figure B.10: Example Console Output of running Marabou-Win



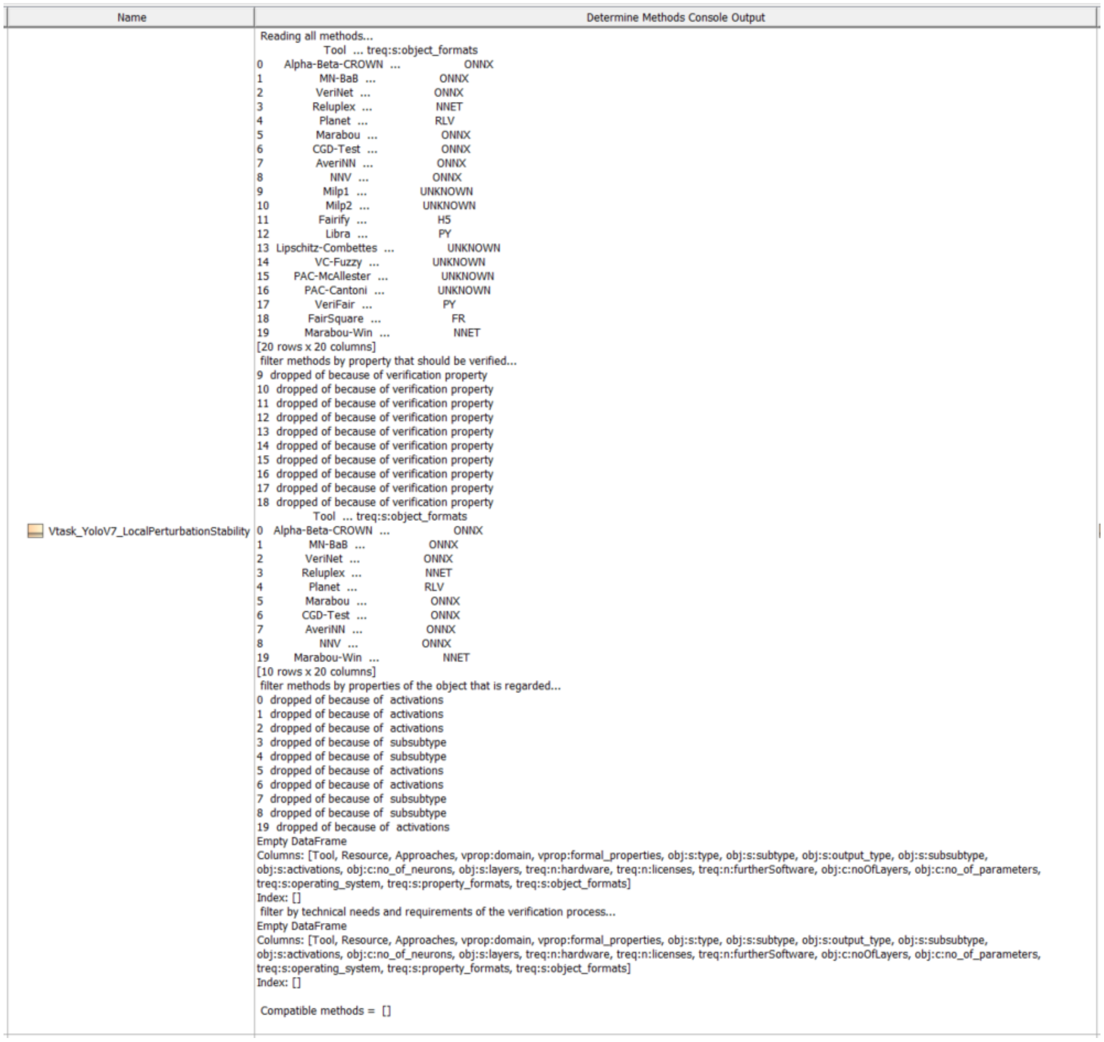
```
true    Unstable pivots performed anyway: 0
        --- Tableau Statistics ---
        Total number of pivots performed: 0
                Real pivots: 0. Degenerate: 0 (0.00%)
                Degenerate pivots by request (e.g., to fix a PL constraint): 0 (0.00%)
                Average time per pivot: 0.00 milli
        Total number of fake pivots performed: 0
        Total number of rows added: 0. Number of merged columns: 0
        Current tableau dimensions: M = 609, N = 1449
        --- SMT Core Statistics ---
        Total depth is 0. Total visited states: 1. Number of splits: 0. Number of pops: 0
        Max stack depth: 0
        --- Bound Tightening Statistics ---
        Number of tightened bounds: 808.
                Number of rows examined by row tightener: 0. Consequent tightenings: 0
                Number of explicit basis matrices examined by row tightener: 0. Consequent tightenings: 0
                Number of bound tightening rounds on the entire constraint matrix: 0. Consequent tightenings: 0
                Number of bound notifications sent to PL constraints: 722. Tightenings proposed: 0
        --- Basis Factorization statistics ---
        Number of basis refactorizations: 2
        --- Projected Steepest Edge Statistics ---
        Number of iterations: 0.
        Number of resets to reference space: 1. Avg. iterations per reset: 0
        --- SBT ---
        Number of tightened bounds: 808
---
22:07:17 Statistics update:
        --- Time Statistics ---
        Total time elapsed: 5614 milli (00:00:05)
                Main loop: 5500 milli (00:00:05)
                Preprocessing time: 89 milli (00:00:00)
                Unknown: 24 milli (00:00:00)
        Breakdown for main loop:
                [36.10%] Simplex steps: 1985 milli
                [50.10%] Explicit-basis bound tightening: 2755 milli
                [0.21%] Constraint-matrix bound tightening: 11 milli
                [0.53%] Degradation checking: 29 milli
                [0.00%] Precision restoration: 0 milli
                [0.03%] Statistics handling: 1 milli
                [1.49%] Consots by request (e.g., to fix a PL constraint): 468 (94.35%)
                Average time per pivot: 0.07 milli
        Total number of fake pivots performed: 33
        Total number of rows added: 0. Number of merged columns: 0
        Current tableau dimensions: M = 609, N = 1449
        --- SMT Core Statistics ---
        Total depth is 3. Total visited states: 44. Number of splits: 21. Number of pops: 22
        Max stack depth: 7
        --- Bound Tightening Statistics ---
        Number of tightened bounds: 31207.
                Number of rows examined by row tightener: 11249. Consequent tightenings: 234
                Number of explicit basis matrices examined by row tightener: 144. Consequent tightenings: 4536
                Number of bound tightening rounds on the entire constraint matrix: 8. Consequent tightenings: 2656
                Number of bound notifications sent to PL constraints: 26137. Tightenings proposed: 4801
        --- Basis Factorization statistics ---
        Number of basis refactorizations: 120
        --- Projected Steepest Edge Statistics ---
        Number of iterations: 11282.
        Number of resets to reference space: 27. Avg. iterations per reset: 417
        --- SBT ---
        Number of tightened bounds: 23560
unsat
```

Figure B.11: Example Console Output of running Marabou-Win



Name	Obj Applied On	Domain	Subdomain	Formal_class	Describtion
▤ LocalTranslationStability	▤ YoloV7	Robustness	Local_Robustness	Local-Input-Output-Property	If the input picture is a translated version of a input picture in the training data, it should get the same label.
▤ LocalPerturbationStability	▤ YoloV7	Robustness	Local_Robustness	Local-Input-Output-Property	If there are small perturbation in the values of the pixels, the image should get the same label.
▤ LocalPhotometricStability	▤ YoloV7	Robustness	Local_Robustness	Local-Input-Output-Property	If the input picture is a photometric transformated version of a input picture in the training data, it should get the same label.
▤ PAC-BayesGeneralization	▤ YoloV7	Generalization	Generalization_Gap_Bound	Probabilistic-Property	The PAC-Bayes bounds of the generalization gap should be small enough.
▤ LocalSubsamplingStability	▤ YoloV7	Robustness	Local_Robustness	Local-Input-Output-Property	If the input picture is a subsampled version of a input picture in the training data, it should get the same label.

Figure B.12: Property table of the threat localization system

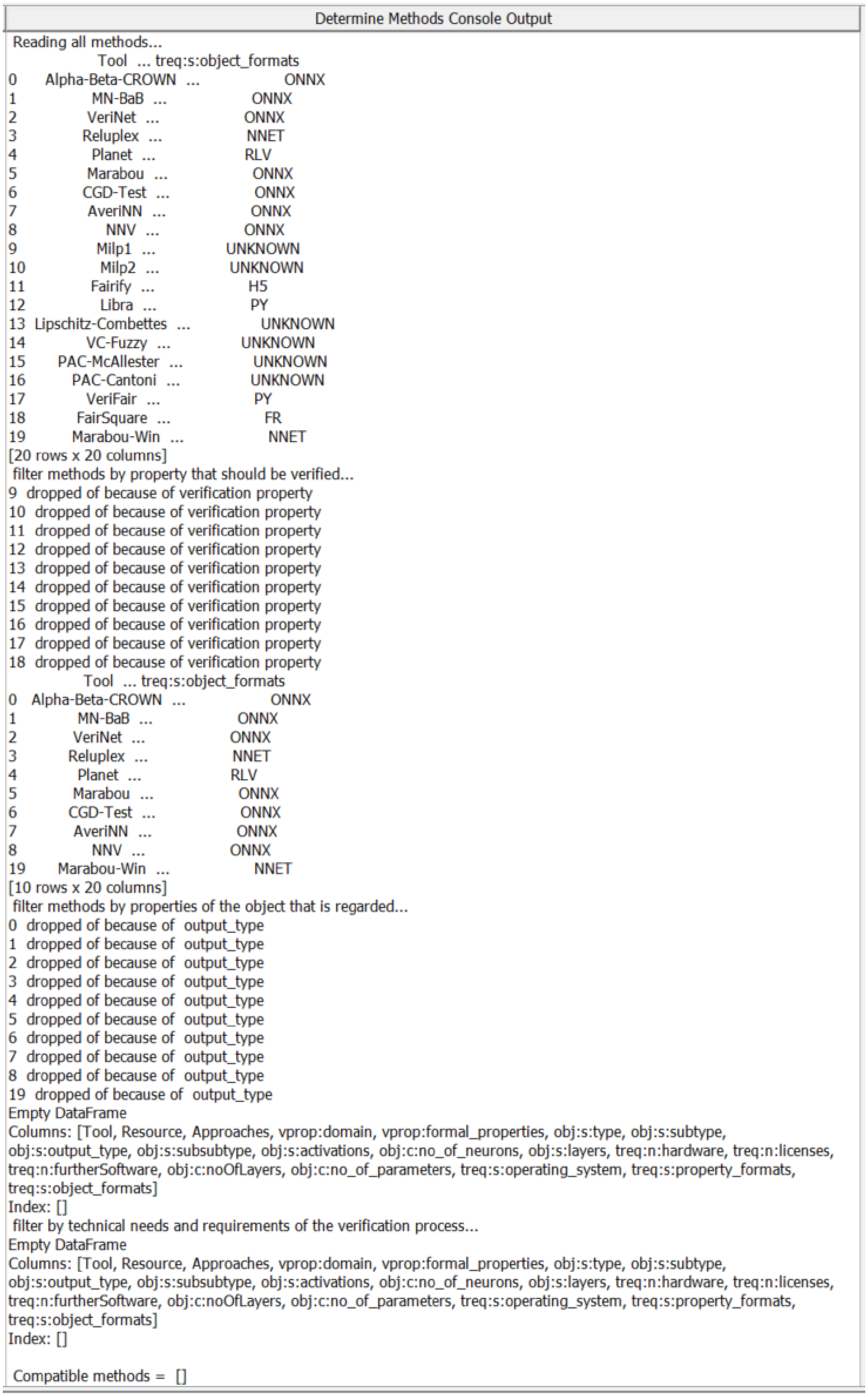


Name	Determine Methods Console Output
Vtask_YoloV7_LocalPerturbationStability	

```
Reading all methods...
              Tool  ... treq:s:object_formats
0     Alpha-Beta-CROWN  ...              ONNX
1              MN-BaB  ...            ONNX
2              VeriNet  ...           ONNX
3             Reluplex  ...           NNET
4               Planet  ...           RLV
5             Marabou  ...             ONNX
6            CGD-Test  ...             ONNX
7              AveriNN  ...           ONNX
8                 NNV  ...           ONNX
9               Milp1  ...        UNKNOWN
10              Milp2  ...        UNKNOWN
11             Fairify  ...            H5
12               Libra  ...            PY
13  Lipschitz-Combettes  ...          UNKNOWN
14            VC-Fuzzy  ...         UNKNOWN
15       PAC-McAllester  ...         UNKNOWN
16         PAC-Cantoni  ...         UNKNOWN
17            VeriFair  ...            PY
18          FairSquare  ...            FR
19         Marabou-Win  ...            NNET
[20 rows x 20 columns]
 filter methods by property that should be verified...
9  dropped of because of verification property
10  dropped of because of verification property
11  dropped of because of verification property
12  dropped of because of verification property
13  dropped of because of verification property
14  dropped of because of verification property
15  dropped of because of verification property
16  dropped of because of verification property
17  dropped of because of verification property
18  dropped of because of verification property
              Tool  ... treq:s:object_formats
0   Alpha-Beta-CROWN  ...              ONNX
1              MN-BaB  ...            ONNX
2              VeriNet  ...           ONNX
3             Reluplex  ...           NNET
4               Planet  ...           RLV
5             Marabou  ...             ONNX
6            CGD-Test  ...             ONNX
7              AveriNN  ...           ONNX
8                 NNV  ...           ONNX
19        Marabou-Win  ...             NNET
[10 rows x 20 columns]
 filter methods by properties of the object that is regarded...
0  dropped of because of  activations
1  dropped of because of  activations
2  dropped of because of  activations
3  dropped of because of  subsubtype
4  dropped of because of  subsubtype
5  dropped of because of  activations
6  dropped of because of  activations
7  dropped of because of  subsubtype
8  dropped of because of  subsubtype
19  dropped of because of  activations
Empty DataFrame
Columns: [Tool, Resource, Approaches, vprop:domain, vprop:formal_properties, obj:s:type, obj:s:subtype, obj:s:output_type, obj:s:subsubtype,
obj:s:activations, obj:c:no_of_neurons, obj:s:layers, treq:n:hardware, treq:n:licenses, treq:n:furtherSoftware, obj:c:noOfLayers, obj:c:no_of_parameters,
treq:s:operating_system, treq:s:property_formats, treq:s:object_formats]
Index: []
 filter by technical needs and requirements of the verification process...
Empty DataFrame
Columns: [Tool, Resource, Approaches, vprop:domain, vprop:formal_properties, obj:s:type, obj:s:subtype, obj:s:output_type, obj:s:subsubtype,
obj:s:activations, obj:c:no_of_neurons, obj:s:layers, treq:n:hardware, treq:n:licenses, treq:n:furtherSoftware, obj:c:noOfLayers, obj:c:no_of_parameters,
treq:s:operating_system, treq:s:property_formats, treq:s:object_formats]
Index: []

 Compatible methods =  []
```

Figure B.13: Example Console Output of running the *DetermineMethod* function on the threat localization system



Determine Methods Console Output

```
Reading all methods...
             Tool  ... treq:s:object_formats
0    Alpha-Beta-CROWN  ...                 ONNX
1              MN-BaB  ...               ONNX
2             VeriNet  ...              ONNX
3            Reluplex  ...              NNET
4              Planet  ...              RLV
5             Marabou  ...               ONNX
6            CGD-Test  ...               ONNX
7             AveriNN  ...               ONNX
8                 NNV  ...              ONNX
9               Milp1  ...           UNKNOWN
10              Milp2  ...           UNKNOWN
11            Fairify  ...              H5
12              Libra  ...              PY
13 Lipschitz-Combettes  ...            UNKNOWN
14           VC-Fuzzy  ...           UNKNOWN
15       PAC-McAllester  ...            UNKNOWN
16         PAC-Cantoni  ...            UNKNOWN
17            VeriFair  ...              PY
18          FairSquare  ...              FR
19         Marabou-Win  ...              NNET
[20 rows x 20 columns]
 filter methods by property that should be verified...
9  dropped of because of verification property
10  dropped of because of verification property
11  dropped of because of verification property
12  dropped of because of verification property
13  dropped of because of verification property
14  dropped of because of verification property
15  dropped of because of verification property
16  dropped of because of verification property
17  dropped of because of verification property
18  dropped of because of verification property
             Tool  ... treq:s:object_formats
0  Alpha-Beta-CROWN  ...                 ONNX
1            MN-BaB  ...               ONNX
2           VeriNet  ...              ONNX
3          Reluplex  ...              NNET
4            Planet  ...              RLV
5           Marabou  ...               ONNX
6          CGD-Test  ...               ONNX
7           AveriNN  ...               ONNX
8               NNV  ...              ONNX
19      Marabou-Win  ...              NNET
[10 rows x 20 columns]
 filter methods by properties of the object that is regarded...
0  dropped of because of  output_type
1  dropped of because of  output_type
2  dropped of because of  output_type
3  dropped of because of  output_type
4  dropped of because of  output_type
5  dropped of because of  output_type
6  dropped of because of  output_type
7  dropped of because of  output_type
8  dropped of because of  output_type
19  dropped of because of  output_type
Empty DataFrame
Columns: [Tool, Resource, Approaches, vprop:domain, vprop:formal_properties, obj:s:type, obj:s:subtype,
obj:s:output_type, obj:s:subsubtype, obj:s:activations, obj:c:no_of_neurons, obj:s:layers, treq:n:hardware, treq:n:licenses,
treq:n:furtherSoftware, obj:c:noOfLayers, obj:c:no_of_parameters, treq:s:operating_system, treq:s:property_formats,
treq:s:object_formats]
Index: []
 filter by technical needs and requirements of the verification process...
Empty DataFrame
Columns: [Tool, Resource, Approaches, vprop:domain, vprop:formal_properties, obj:s:type, obj:s:subtype,
obj:s:output_type, obj:s:subsubtype, obj:s:activations, obj:c:no_of_neurons, obj:s:layers, treq:n:hardware, treq:n:licenses,
treq:n:furtherSoftware, obj:c:noOfLayers, obj:c:no_of_parameters, treq:s:operating_system, treq:s:property_formats,
treq:s:object_formats]
Index: []

 Compatible methods =  []
```

Figure B.14: Example Console Output of running the *DetermineMethod* function on the threat localization system with the attribute *output_type* added