Carnegie Mellon University
Software Engineering Institute

# EXPLAINABLE VERIFICATION: SURVEY, SITUATIONS, AND NEW IDEAS

*Bjorn Andersson*
*Dionisio de Niz*
*Mark Klein*

November 2023
[DISTRIBUTION STATEMENT A]

## Abstract

Many software-intensive systems of current and future application domains require (or will require) approval from a certification authority before being deployed. Formal methods (FMs) verify properties of software by proving them mathematically rather than running tests of the software. FMs have the potential to help certification. But unfortunately, FMs are currently rarely used because software practitioners do not understand how FM tools work internally and do not understand the underlying theories that these tools depend on. Since it is hard to trust something that you do not understand and have not used before, software practitioners tend to not trust FM tools. If FM tools could explain their output, however, then software practitioners would be more likely to trust them and this could bring the benefits of (i) faster fielding (because of less testing), and (ii) improved safety (because of better coverage).

Therefore, this report focuses on potential changes in software development practice and the research needed for this to be realized. Specifically, it discusses explanations generally, it discusses the way explanations can be used in different types of verification, and it presents ideas for creating explanations for FMs.

## 1 Introduction

### 1.1 Background

Many software-intensive systems of current and future application domains require (or will require) approval from a certification authority before being deployed. Examples of such application domains include: aircraft, medical devices, spacecraft, autonomous ground vehicles, autonomous air vehicles. Examples of current certification authorities include: Federal Aviation Administration (FAA), European Union Aviation Safety Agency (EASA), Food and Drug Administration (FDA).

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

## 1.2 Current pain

Today, many application domains have a set of guidance documents. For example, in civil aviation regulated by the FAA, the document DO-178C [DO-178C 2011] provides the most common means for approval of software. These guidance documents tend to be process-oriented; i.e., (i) they prescribe how the development of the system should proceed, (ii) they prescribe how the applicant (the organization that develops the system) should communicate with the certification authority, (iii) they state high-level objectives, and (iv) they state pitfalls that should be avoided. This mindset has been successful in many domains. For example, among US air carriers, the safety record today is much better than it was decades ago. Unfortunately, this mindset also has some limitations. These include: (i) limitations for future application domains, (ii) limitations on permitting frequent and late changes, (iii) limitations on being process-driven rather than focusing on direct evidence of the safety of the software, and (iv) not taking full advantage of the research within the formal methods and the real-time systems research community: the knowledge of these communities is not present in these documents and these documents do not cite papers from these research communities.[1] Achieving safety through extensive testing appears to be problematic because it precludes frequent and late changes. Achieving safety through models fed into verification techniques requires tool qualification (for aircraft, see DO-330 which supports DO-178C [DO-178C 2011]; for automotive, see page 30 of Part 8 of ISO 26262 [ISO26262part8 2018]). Thus, it is worth exploring alternatives, specifically exploring (i) what explainability means, (ii) whether explainability can help, and (iii) how it can be achieved for formal methods.

## 1.3 Goal

The goals of this report are to

G1. survey the state-of-the-art in explainability in various disciplines.

G2. describe situations where explainability is useful.

G3. generate ideas that can be useful for explainability of formal methods and real-time systems.

G1 and G2 are useful for (academic) research communities of real-time systems and formal methods. G3 might be useful for these communities as well. But it is certainly useful for an SEI internal research agenda.

Regarding G1, G2, and G3, it is worth noting that explainability of formal methods and real-time systems is a new research area and it is important to shape it. There is currently no survey and there is currently no document that lists ideas for this.

---

[1] A recent survey [Garavel 2020] of practitioners indicate that most believe that FMs will spread more widely in industry (see Section 7.1 in [Garavel 2020]) and they will not be overshadowed by other technologies (see Section 7.4 in [Garavel 2020]).

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

Regarding G3, it is worth noting that (i) SEI has funded projects on explainability of computer vision [SEIGreybook 2022] and robots [Rosenthal 2016], and (ii) SEI researchers have advocated for bi-directional explainability in Cyber Security [Mellon 2023]. But these are different from our focus, which is on explainable verification. Also, there are many academic research efforts focusing on explainability of artificial intelligence (AI) but, once again, this is different from our focus.

For these reasons, we believe goals G1, G2, and G3 are worth pursuing.

## 1.4 Terminology

The term FMs can have different meanings in different contexts. In the area of safety, a hazard analysis like failure mode and effect analysis (FMEA) is often labelled as an FM because it is executed formally through worksheets (with guidewords) by a human. However, in academic research in computer science, this would not be viewed as an FM. Instead, an FM is a method that uses automation (i.e., the work is done by a computer) to reach a conclusion. Typically, such FMs use either (i) deduction, (ii) searching, or (iii) experiments that show that a certain behavior is possible. We are primarily interested in the meaning used in academic research in computer science.

Also, the word verification can have different meanings in different contexts. A human performing inspection of source code is sometimes referred to as verification. However, we are primarily interested in verification that is performed automatically (i.e., the work is done by a computer).

Some software practitioners use the word software verification to mean an activity related to a design artifact; for example (i) checking source code to determine whether it complies with a certain coding convention (e.g., MISRA C), (ii) checking compiler warnings, (iii) checking source code to determine whether it is consistent with architectural documents, or (iv) checking source code to determine whether it traces to requirements. However, in academic research in computer science, this would not be viewed as a verification. Instead, academic research in computer science tends to view verification to mean the following: given a design artifact (e.g., source code), the design artifact can generate different behaviors (e.g., execution traces, sequences of state transitions, schedules) and we are interested in determining whether a certain property is true for all behaviors (or for all behaviors that satisfy some restriction that we impose).

# 2 State-of-the-art in explainability

The area of explainable verification is in its infancy. This section starts by surveying the research literature in explainability in other areas than explainable verification. Then, it surveys recent work in explainable verification.

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

## 2.1 Explainability not related to software

Explainability has been studied in many disciplines. In philosophy, in ancient Greece, Aristotle studied causality as part of his four forces in his theory of forms; he also made the distinction between "demonstration of the fact" and "demonstration of the reasoned fact," the latter being an explanation. Many philosophers have invoked the "Principle of sufficient reason" as claiming that for each proposition or event, causality exists [Wikipedia 2024]. Later Craik [Craik 1943] introduced the notion of a mental model as an explanation. Lipton [Lipton 2001] viewed explanation as understanding and presented five concepts of understanding, argued their respective merits, and argued that the concept of causality is the best of them. Lipton proposed the notion of contrastive explanation; instead of answering "why X" it explains "Why X rather than Y?" Deutsch [Deutsch 1997,Deutsch 2009,Deutsch 2012] argues that (i) explanation is more important than prediction in theory of science, and (ii) a good explanation should be hard to vary; i.e., if we change an element in an explanation, then the explanation should be false. David Deutsch also points out that Darwin's theory of evolution provides explanation rather than prediction. As an example, he states that Darwin's theory does not predict the existence of elephants but it explains the existence of elephants [Deutsch 2023]. In psychology, Keil [Keil 2006] points out that (i) an older view was that all explanations are deductive proofs (DN model) but this has not survived well, (ii) explanations have different purposes (assigning blame, making predictions for the future, finding a root cause of an event), (iii) explanations can be categorized as mechanistic, design, and intent, and (iv) all cultures use the same type of explanations but they vary in which explanation they emphasize. It has been argued [Wikipedia 2023a] (as part of the movement on pragmatic explanations within the area Scientific Explanation) that an explanation should use concepts that are in the mind of the person receiving the explanation; this was a critique against the DN model and brought the pragmatic model of explanation. Metaphysical explanation [Wikipedia 2023b] considers explanations of statements x is X because there is an explanation of x that is derived from the definition or essence of X. Mathematical explanation [Wikipedia 2023b] focuses on whether proofs are intended to establish truth or provide understanding. In mathematical logic, if a formula $A(x,y) \wedge B(y,z)$ is unsatisfiable, then a Robinson interpolation [Hinman 2005, page 66] is a formula that explains the unsatisfiability and contains only variables that A and B share. Bermúdez [Bermúdez 2020] provides an introduction to cognitive science. It focuses on minds (both human and animals) but goes beyond behavioralism; it studies mental processes (not just input-output behavior). Many of the results point to limitations in human information processing and how minds take short cuts (e.g., limitations on working memory, filtering of information) and learning with conditioning. Human-Centered Design [Norman 2013] studies how the human mind interacts with products; pointing out [Norman 2013, page 25] "A conceptual model is an explanation, usually highly simplified, of how something works" and [Norman 2013, page 52] that users have expectations on behavior of products and when a product gives feedback to a user, a user feels more in control; this gives rise to positive emotion. The importance of giving clues to users is also emphasized in Krug [Krug 2010, Krug 2014].

## 2.2 Explainability related to software

In software testing, a common technique in safety-critical systems is MC/DC testing [Wikipedia 2023d]; that is, creating a test suite so that each decision in a program is covered and each condition in

a decision can affect the outcome. The latter can be viewed as contrastive explanation. In constraint satisfaction, explainability has been considered [Gupta 2021] in terms of explaining why a set of constraints are unsatisfiable. In computer science, a regular expression can be thought of as an explanation for the set of strings that a finite-state automaton accepts and given a small number of strings, one can learn an automaton [Angluin 1987]. Pearl [Pearl 2019] points out that a joint probability distribution can be represented by a graph where each node represents a random variable and an edge represents a dependency. In this way, it is possible to model causality and consequences of intervention; this can be viewed as explanation. The area of safety can be thought of as: hazard analysis and assurance case. In the former [Ericson 2016], it is common to scan guidewords (e.g., energetic material) in design documents to find a hazard (a condition within a system so that if a condition outside the system is unfortunate, then a mishap can occur). The latter [Kelly 1998,Diemert 2023] involves constructing a tree of claims so that the top-level claim is what we care about (the entire system is safe) and for each non-leaf claim, it holds that it can be justified based on its children claims. Both can be viewed as providing explanation (of mishap or safety). The standard DO-178C [DO-178C 2011, Hilderman 2021] relies on tracing requirements; such tracing can be viewed as explanations. The area of statistics provides techniques for exploratory data analysis; for example, reducing a dataset to lower dimension so it can be visualized while maintaining some important information about the original dataset [Fridman 1974]; the lower dimension representation can be used when explaining phenomena in the original dataset.

## 2.3 Explainability related to artificial intelligence

Statistics and Machine Learning (ML) have traditionally had different goals. The latter aims for high prediction performance with black boxes; the former starts with a structure and aims for understanding. ML belongs to the broader area of Artificial Intelligence (AI). Explainable AI (XAI) has become important because of the desire to use AI in high-stakes decision making. The DARPA XAI program [DARPA 2022] studies this and has provided a categorization of XAI in terms of (i) deep explanation, (ii) interpretable models, and (iii) model induction. Hagras [Hagras 2018] argues in favor of using fuzzy logic for XAI. Holzinger [Holzinger 2018] provides a survey of XAI techniques with emphasis on its use in healthcare. Hoffman [Hoffman 2018] studies how to evaluate explainability. Falco et al. [Falco 2021] discusses government regulation of XAI.

Details on explainability are discussed below for both the systems based on large language models (LLM), reinforcement learning (RL), and for those without. It is noteworthy that the words *explainable* and *interpretable* are often used as synonyms (also pointed out in [Zhao 2023]). Also, an explanation is said to be local if it applies to a specific input to a function; an explanation is said to be global if it applies to all inputs to a function.

### 2.3.1 Non-LLM and Non-RL work

Examples of high-profile works in explainable AI are:

1. Local Interpretable Model-agnostic Explanations (LIME) [Ribeiro 2016]

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

2. Class Activation Mapping (CAM) [Zhou 2016]
3. Gradient-weighted Class Activation mapping (Grad-CAM) [Ramaprasaath 2017]
4. Layer-Wise Relevance Propagation (LRP) [Bach 2015]
5. SHapley Additive exPlanation (SHAP) [Lundberg 2017]
6. Deep Learning Important FeaTures (DeepLIFT) [Shrikuma 2017]
7. Optimal Classification Trees [Bertsimas 2017]

These works are discussed below. We choose to discuss them because they are highly cited. The following is common among the aforementioned works: (i) they rely on the idea that a human has selected a set of features, and the explanation method explains the current mapping based on those features, and (ii) they focus on the use of ML in health care.

#### 2.3.1.1 LIME and SP-LIME

LIME and SP-LIME [Ribeiro 2016] are a model-agnostic approach for interpretability of a ML system. LIME is a basic version for one type of interpretability (local); SP-LIME is a variation of LIME and it performs another type of interpretability (global). Both assume supervised learning but make no assumption on training data, inductive bias, training method, or representation of function. They assume that through training, we already have a function f that maps input to output. Then, these approaches can be applied after training. LIME can be used at inference time and it is run each time we get a new input. SP-LIME can be used at any time after training. We can use SP-LIME when a system is being deployed and its training has been completed but we have not received any inputs yet. We can also use SP-LIME whenever we receive a new input at run-time. LIME solves the problem P1 below. SP-LIME solves the problem P2 below.

- P1. Given a specific input I and a function f that maps I to the output O, explain why the output O was produced based on I.
- P2. Given a function f that maps inputs to outputs, explain approximately what this function does.

An explanation for P1 can help a human to trust the ML system for a particular input I. An explanation for P2 can help a human to trust the ML system in general (not just for one input).

LIME produces an explanation to P1 as follows: One can use any supervised learning technique to train the ML system with training data set. Then, at inference time, when the ML system receives one input I and is expected to produce an output O, LIME is invoked. LIME generates perturbations of the input I and for each such perturbation LIME applies the ML system to get an output. Then, LIME uses these perturbations to learn a simple function (which we call g). The function g produces an output in the same space as f. However, the input to g is different from the input to f. It is assumed that we have a special feature space representation of the input to the ML system. For example, if the input to the ML systems is an image, then a feature representation of this image is a vector where each component in the vector is zero or one; the interpretation is that if the feature is present in the input, then, the

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

value in the feature vector is one. In this example, a feature could be detection of a certain type of edge or line segment. The idea is that when an input is fed into an ML system, we compute the feature-representation of this input. This input is fed to g and it produces an output. The function g should be a linear function of the feature representation and ideally the output produced by g should be approximately equal to the output produced by f. We should choose the feature representation of an input so that (i) the feature representation is a vector, (ii) the dimensionality of the vector is permitted to be large but it should hold for most inputs that in the feature representation, most components are zero (i.e., only a small number of components are one). When training g, we use a special type of loss function (minimize sum of magnitude of weights) that gives us that the coefficient that represents g is sparse. We can now interpret f for an input. Given an input I, we create perturbations of I; then compute the feature representation of I and the feature representation of all perturbations of I; then train the function g; then identify the coefficients in g that have the largest magnitude; then identify the components in the feature representation that corresponds to those coefficients; then report those components in the feature representation to the user and state that "these features are responsible for the output for function f for this input."

SP-LIME produces an explanation to P2 as follows: Generate a set of random inputs to the ML system. For each input, obtain an explanation using LIME. Now, we have a set of inputs to the ML and for each input, we have an explanation. Then, select a set of features as follows. For feature j, compute $I_j$ such that $I_j$ is large if the weight of j is large in many of the explanations. Then, select the features with large values of $I_j$. Then report those features to the users and state that "these features are responsible for the output for function f for the input space."

#### 2.3.1.2 CAM

CAM [Zhou 2016] is not focused on explainability per se but can support it and has been used later for explainability. CAM considers an artificial neural network (ANN) that performs image classification. The ANN is given as input an image and the ANN is also designed considering a set of categories (e.g., "dome' "palace" "church") and the ANN is supposed to determine which category is in the image. CAM works as follows. The ANN has a set of layers that are convolution layers; these detect features in the image (with earlier layers detecting simpler features). Such an ANN is called a convolutional neural network (CNN). It is assumed that the system has no fully connected layers. When such a classifier is designed for CAM, the system does not just output the category that is in the image; it also outputs a CAM where a CAM is marking for pixels (actually, pixels in the last convolutionl layer) that states how important this pixel is for this category. For example, if an image classifier is given an image of a dome, then the classifier will output a marking that shows which pixels are important for determining that the image is a dome.

#### 2.3.1.3 Grad-CAM

Grad-CAM [Ramaprasaath 2017] considers a CNN that performs image classification and it considers explainability of the output of the CNN. Grad-CAM extends CAM so that it also allows fully connected layers; hence Grad-CAM has the potential to achieve better prediction performance than CAM. The paper [Ramaprasaath 2017] also emphasizes explainability. It also shows how to obtain a set of

pixels that serve as counterfactual explanation; that is, changing these pixels will cause the classifier to output another category.

#### 2.3.1.4 LRP

LRP [Bach 2015] solves the same problem as CAM and Grad-CAM but it differs in that for LRP, when an artificial neural network is given an image, LRP assigns a relevance metric for each neuron in each layer (that is how much it influences the output).

#### 2.3.1.5 SHAP

Many explanations select a set of features; thus, it is necessary to quantify the goodness of the selection of features. SHAP [Lundberg 2017] uses an idea from cooperative game theory to select features. In cooperative game theory, one considers n players and each player decides whether to participate in a coalition. There is a function that maps for each set of players a utility that is accrued if this set of players form a coalition. The utility needs to be distributed among the members of the coalition. The literature provides different constraints on this distribution and different methods for computing the utility to each player in the coalition so that this distribution satisfies constraints. There is one set of constraints that yields a unique distribution of utility among the participants in the coalition. By substituting a player for a feature in ML and substituting a coalition for a selected set of features in ML, we obtain a method for selecting features. SHAP uses this idea to select a set of features that explains the output of an ML system for a given input.

#### 2.3.1.6 DeepLIFT

DeepLIFT [Shrikuma 2017] creates an explanation similarly to LIME and has an explanation vector x’ associated with each input x (just like LIME) but in DeepLIFT, the 0-1 values in x’ have different meaning. In DeepLIFT, if a component in vector x’ is 1, then it means that x’ takes its value from a component in x; but if the component in vector x’ is 0, then it means that x’ takes its value from a reference value.

#### 2.3.1.7 Optimal Classification Trees

Bertsimas and Dunn [Bertsimas 2017] argue that decision trees are more explainable than other types of machine learning; it is easy for human users to just inspect the decision nodes and if a human feeds an ML system with input x, then it is easy to see which path it passes through the tree. However, this is only true if the decision tree is sufficiently small. Therefore, Bertsimas and Dunn ask whether it is possible to find the smallest decision tree. Previous work on decision tree has used heuristics but Bertsimas and Dunn improve on that by formulating the construction of a decision tree as a constraint optimization problem (linear constraints with some variables having the domain integer). They argue that there have been large improvements in performance in solvers (e.g., Gurobi) so that such construction is feasible today; they show this with extensive experiments.

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

### 2.3.2 Explainable RL

Maduma [Maduma 2019] presents explainable RL. Specifically, given an agent, and knowledge of how the agent's action influences the environment, explain the agent's behavior. It can explain why the agent took action A rather than not A.

### 2.3.3 LLM work

Zhao [Zhao 2023] provides a survey of explainability of LLMs. There are different types of explanations, discussed below.

Feature-attribution-based explanations aim to measure the relevance of an input feature (e.g., a word) to the prediction. There are many sub-types of them. Perturbation-based explanation is one such example. It perturbs input and observes how these perturbations change output. The aim is to find small changes in input that leads to a change in output. Gradient-based explanation works similarly but instead of relying on perturbations to find how input changes output, it computes the derivative of the output with respect to inputs. Surrogate models use simple models (that are easy for humans to understand) to give similar predictions as the original predictor (which is a black box). Decomposition methods break down output as a function of input.

Attention-based explanation explain behavior based on identifying relevant parts of the input. Visualization methods visualize attention heads (which are part of the transformer architecture that is used in LLMs) for a specific input. Function-based explanation considers the partial derivative of the output as a function of attention weights (not the inputs).

Example-based explanations include one type of explanations called counterfactual explanation. It considers the case that given an observed input x and a perturbed input x' with certain features changed, the prediction y would change to y'. Typical perturbations are paraphrasing or word replacement.

Neuronal activation explanations identify individual neurons that are crucial for performance. It can involve two steps: (i) identify important neurons in unsupervised manner, and (ii) learn relations between linguistic properties and individual neurons in supervised tasks.

The survey [Zhao 2023] points out that one use of explanations is in debugging. For example, if the machine learning model consistently attends to certain tokens in the input sequence regardless of the context, this may indicate that the machine learning model relies on heuristics or biases rather than truly understanding the meaning of the input sequence.

### 2.3.4 Sparsify

Since ANNs tend to have more weights than training examples, it is common to apply regularization techniques to training so as to avoid overfit. L1 regularization is a common one. It introduces an extra penalty term in the loss function used for training. This extra penalty is the sum of magnitude of the weights. Clearly, this encourages the training to use weights with smaller magnitude. But it also encourages the training to sparsify the ANN; that is, assign zero to many of the weights and assign non-

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

zero to a small number of weights. For such an ANN, we can remove all connections that have weight zero. This makes the ANN simpler and more interpretable.

Recently, Liu et al. extended this idea to improve interpretability further [Liu 2023]. The idea is to embed each neuron in a 2-dimensional space. The $j^{th}$ neuron in the $i^{th}$ layer is placed at coordinate (i,j). Then, the training encourages locality. It is achieved as follows. Pairs of neurons that are placed far apart in the coordinate system are discouraged from having large weights. And they are discouraged from having non-zero weights (that is, they are encouraged to have zero weighs). In this way, after training, most of the connections between neurons are local in the sense that for the j:th neuron in the $i^{th}$ layer, if it is connected to a neuron in the $i+1^{th}$ layer, it is more likely to be connected to the $j^{th}$ neuron (same row) than some other neuron in the $i+1^{th}$ layer. Hence, the ANN can be thought of as a set of modules where the interactions between modules is much smaller than the interactions within a module. By identifying the modules "visually" one can "see" how the learned function works from its parts. This further improves interpretability.

A very recent results by Michaud et al. [Michaud 2024] has used sparsity and several other ideas to achieve interpretability. They consider a recursive neural network (RNN). An RNN is an ANN such that the some of the inputs to the ANN are output signals from the ANN; this is feedback and these signals can be thought of as a state. Their idea to achieve interpretability is to decode a given RNN into a finite state machine represented as a Python program. Their idea is to run the RNN several times and record the states that are reached. Then, one forms clusters so that for each cluster, there are a number of states that that are similar. Then, one hypothesizes that there is a set of discrete program variables (Booleans, integers) in a Python program such that if one assigns values to these discrete program variables, then there is a mapping from these values to a cluster. Michaud et al. [Michaud 2024] present methods to find such mappings. This requires enumerating hypotheses about program variables, and it requires searching through many (or all) assignments of values to program variables; but it does not require searching through the set of all Python programs. They also present methods to find such mapping with very little searching for the case that there is regularity in the clusters. Given this, the only challenge that remains is to find a function f and a function g such that (i) f represents the change of values of the program variables for each computation step in the Python program, and (ii) g represents the output produced based on the state. Finding f and g can be achieved using symbolic regression. This yields a Python program that represents a finite state machine that behaves like the given RNN.

#### 2.3.5 Explainability of OpenAI's transformers

The corporation OpenAI has made public a tool for debugging transformers—see [TDB 2024].

## 2.4 Explainability related to formal methods

With respect to FMs, explainability has been recognized by a small (but growing) number of authors as important because many FM tools have been found to be defective (produces wrong output). This has been the case in SAT/SMT solvers which are the foundations of model checking tools [Mansur 2020, Winterer 2020, Park 2021], it has been the case in model checking tools [Zhang 2019], and it

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

has been the case in timing analysis tools [Davis 2007] (more examples are given in Section 2 in [Maida 2022]). CBMC [Biere 1999] (one of the common model checkers) can produce a counterexample but it is difficult for humans to understand a counterexample [Martins 2022] and there is no explanation for the case that the property holds. The SLAM model checker [Ball 2004, Ball 2011] (which is based on forming an abstract state machine using predicate abstraction of source code) can provide a counterexample; specifically, it provides (as stated on page 5 in [Ball 2011]) "a possible execution trace through the driver that shows how the rule can be violated" but no explanation for the case that the property is true. [Chechik 2005] considered explanations of counterexamples and presented two ideas: (i) creating a proof that a state in the counterexample can be reached, and (ii) abstraction of counterexample. In the authors opinion, these explanations are quite hard to understand.

In model checking, it is common to distinguish between safety properties and liveness property where (i) in the former, the property states that nothing bad will happens (i.e., the system does not enter an error state), and (ii) in the latter, the property states that eventually something good happens (e.g., if a process requests a mutually exclusive resource, then eventually it will get it). If a safety property is false, then a tool may produce an execution in which the property is violated. This can be presented to a user, as discussed above. If a liveness property is false, then a tool may find a cycle of states (often called "lasso") so that the execution is forced to execute in this cycle and hence never reached the desired good state that we would like to eventually reach. Such a cycle can be thought of as an explanation too.

In recent years, some FMs improve simply because they rely on a SAT/SMT solver and the SAT/SMT solver improves. But FMs in themselves have also developed during recent decades. Typically, this involves (i) new ways of describing a system and correctness properties, (ii) new ways of reasoning about the behavior of a system, (iii) new types of logic applied, (iv) new types of abstraction used, and (v) applying results that were known in logic but has hitherto not yet been used in FM. For example, in terms of theorem proving, a Hoare triple can describe that if a precondition P is true and one part of a program S executes, then a post-condition Q is true. These can describe smaller parts of a program and then be combined to describe larger parts of a program so that eventually, they describe the entire program. In terms of model checking, a common approach has been to compute the set of reachable states and check if it contains an error state. Schedulability analysis views FMs from a high-level abstraction, describing the system as a set of concurrently executing tasks where each task is described with parameters and one computes the cumulative amount of time that a task can execute in various time intervals of given duration. Network calculus has been used to analyze buffers. Lyapunov stability criterion has been used to prove stability of a controller together with its plant. Formal verification became more visible with the SLAM engine which was part of the Microsoft Static Driver Verification [Ball 2004,Ball 2011]; it is based on predicate abstraction (PA) [Graf 1997,Clarke 2018] which performs model checking on a coarse-grained state space and refines it as needed. Despite the advances in formal verification, there has been very little focus on their explainability. We have recently organized a workshop (ERSA) and seen an increasing interest in explainability [ERSA 2022, ERSA 2023].

Z3, one of the most popular SMT solvers, provide two features "proof mining" and "tracing." It is possible that these can help to provide explanations of why Z3 produced a certain output ("satisfiable"

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

or “unsatisfiable”) and it is possible that such explanations can also be used by tools that rely on SMT solvers.

Recently Kaleeswaran [Kaleeswaran 2023] has written a PhD thesis on the topic of creating explanations for counterexamples output from a model checker. He focuses on processing the counterexample (e.g., shortening it, removing states that are not relevant) and on “lifting it” so that the explanation is not about an execution trace (sequence of states) on a low level but uses concepts of higher levels (components). He gives a design flow where (i) one has a design model (e.g., UML diagram) from which one generates a verification model, (ii) one has requirements given in natural language from which one generates a specification (i.e., correctness conditions), (iii) the verification model and specification are fed into a model checker, (iv) if the model checker outputs a counterexample, then this counterexample is fed into the explanation generator, (v) the explanation generator processes the counterexample (e.g., shortening it and/or removing irrelevant states), (vi) one creates a representation of the counterexample (graphical, textual, tabular, trace), and (vii) a human interprets the counterexample by looking at its interpretation. Kaleeswaran conducted a literature survey [page 29-40, Kaleeswaran 2023] and found that:

1. Fault-trees tend to be easier for users to understand that execution traces.
2. Animations are highly effective for user’s comprehension.
3. Shorter counterexamples tend to be easier to understand.
4. In probabilistic model checking where one wants to show that the probability of an error is greater than some bound, then an explanation can be a set of execution traces so that the sum of probability over these execution traces exceeds the bound.
5. If one has a counterexample, then it can be helpful to also have a witness trace in which the correctness property is true; thus the user can compare the counterexample with the witness trace and gain understanding.
6. In some cases, just visualizing each step of a counterexample may not be sufficient for human understanding.
7. Finding causality of an error may be helpful.
8. Using a domain-specific vocabulary may improve comprehension.
9. It is helpful to also create explanations for incomplete input.
10. A gap in the literature is explanations that considers that the audience of an explanation may lack expertise. There are two types of lack of expertise: (i) lack of expertise in formal methods, and (ii) lack of expertise in the application domain.
11. A gap in the literature is to create explanations that can “localize the fault precisely within the specifications and components of the design.”

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

### 2.4.1 Potential use of Lagrangian relaxation

Many formal methods involve proving an upper bound on a quantity (e.g., response time or a program variable). We may want to search for a number that is an upper bound on this quantity and then have a simple proof that this number is indeed an upper bound. The idea of a Lagrangian relaxation can potentially serve this purpose. It works as follows. Consider an optimization where we want to maximize an objective function subject to a set of constraints. Form two partitions of the set of constraints. Create a new optimization problem by keeping the first partition of constraints and removing second partition of constraints. Then, modify the objective function by adding a new term so that violation of the second partition of constraints decreases this term (penalty). Typically, this term includes a set of additional variables called $\lambda_1$, $\lambda_2$,… Set it up so that for any assignment of non-negative values to $\lambda_1$, $\lambda_2$,… it holds that the value of the objective function of an optimal solution for the new problem is an upper bound on the value of the objective function of an optimal solution for the original problem. Hence, an assignment of values to $\lambda_1$, $\lambda_2$,… can be thought of as an explanation for an upper bound of the original problem. Note that different values assigned can yield different tightness of these bounds.

## 2.5 Verification of AI

The area of verification of AI does not aim to produce human-understandable explanations. However, we list some results from this area here because (i) this area is related to explanations in the sense that it aims to prove a property of an AI system, and (ii) this area provides ideas that may be useful for later works on explainability of FM tools. The results listed in this section are all about artificial neural networks (ANNs).

### 2.5.1 ANNs are hard to analyze

It is known that ANNs are hard to analyze [Szegedy 2013, Goodfellow 2015].

Szegedy [Szegedy 2013] makes two important remarks regarding image classifiers. First, it is tempting to view the output from a neuron in the last layer as a basis function that represents a feature in the input space and that this basis function (feature) has semantic information and that the last layer combines these features. However, paper [Szegedy 2013] points out that this view is inappropriate. Second, paper Szegedy [Szegedy 2013] points out that there are many adversarial examples. An adversarial example is an input x' that is very close to another input x such that x and x' output different categories; yet the difference between x and x' are imperceptible to a human. These adversarial examples persist across training methods, hyperparameters used for training, and ANN architectures.

Goodfellow [Goodfellow 2015] points out that the cause of adversarial examples is linearity. To see this, consider a single training example and an ANN with a large number of weights. Compute the loss function used during training for this training example. Then, compute the gradient of this loss function with respect to input to the ANN. This gives us the direction in the input at which the loss function increases the most. This direction is a vector. We can multiply this vector by a scalar $\varepsilon$ (which is a small number) and this yields a small perturbation on this single training example so that the loss function increases by an amount that is the inner product of the perturbation and the gradient; this can

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

become a large amount. Even if ε is small, this increase in the loss function can be large because the number of weights is large and hence the two vectors that we multiply have large dimension. Goodfellow [Goodfellow 2015] finds that (i) a training procedure can be created that mitigates adversarial examples by adding a new example to each of the original examples and these added examples are created using the insight above, (ii) Radial Basis Function (RBF) networks resist adversarial examples, (iii) adversarial examples exist in broad subspaces [page 7, Goodfellow 2015], and (iv) ensamples are not resistant to adversarial examples [page 9, Goodfellow 2015].

### 2..5.2 Proving properties of ANNs

For ANNs, an important work has been the Reluplex algorithm [Katz 2017, Karz 2022]. It allows asking questions about a neural network that uses rectified linear unit (ReLU) neurons. The way it works is that one introduces a variable that represents each signal within the ANN. For example, if the ANN takes n inputs, then we have n variables for that. For the 1st layer, one introduces, for each neuron, one variable that represents the output of the neuron. For the 1st layer, one introduces, for each neuron, a variable that represents the signal that is fed into ReLU of the neuron. Then, one also introduces constraints expressing how the output of a neuron depends on the signal that was fed into the ReLU and how this signal depends on the input to the neuron. With these variables and constraints, one can ask questions about the ANN. An example of a question is: for a given neuron in the last layer, can its output be greater than 42? This is achieved by checking whether this constraint satisfaction instance is satisfiable. Reluplex answers this question by (i) reformulating the Simplex algorithm, originally developed for Linear Programming (LP), as a set of rules, and (ii) incorporating these rules and this constraint satisfaction instance in a Satisfiability Modulo Theories (SMT) framework. The original version of Reluplex [Katz 2017, Karz 2022] worked only for ANNs with ReLU neurons but a later version is more general [Katz 2019].

Dvijotham [Dvijotham 2018] studies the same problem of proving a property of the output of a neuron in the last layer. It considers that the property is expressed as a linear combination being less than or equal to a bound. It expresses that we are only interested in proving this for the case that certain restrictions apply to the input. This restriction is a neighborhood of one nominal input (e.g., a single training example). The main idea of paper [Dvijotham 2018] is the following. One can express this as an optimization problem: maximize a linear function of the output neuron of interest subject to constraints that express what the neurons does. If the maximum obtained is less than or equal to a certain bound, then the property is true. The optimization problem is very hard to solve, however. Therefore, paper [Dvijotham 2018] presents a relaxation technique. It applies Lagrange relaxation; that is, some constraints are removed and instead one models their effect by introducing a penalty term in the objective function so that the violation of a constraint makes the objective function worse. The paper [Dvijotham 2018] observes that if we have lower and upper bounds on the variables that express the internal signals of the ANN, then this optimization problem can be rewritten so that it is separable over layers. Hence, we obtain one optimization problem per layer and this optimization problem is easy to solve. Combining the solutions to these optimization problems yields an upper bound on the original problem and this can be used to prove that the original property is true. The paper [Dvijotham 2018] also presents an approach called interval arithmetic that can be used to compute the lower and upper bounds mentioned above.

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

#### 2.5.3 Provably correct ANN through training

There have been works on training ANN to be correct [Gowal 2018, Wong 2018, Ehlers 2017, Lin 2022, Raghunathan 2018].

Gowal [Gowal 2018] considers an ANN used as a multi-class classifier. Gowal [Gowal 2018] uses ideas similar to the ones in paper [Dvijotham 2018] but in addition, paper [Gowal 2018] presents a training procedure to satisfy the property. The idea of the training procedure is as follows. Normal ANN training uses a loss function, and this loss function describes, for a training example, how much the ANN output deviates from the expected output of the training example. The training procedure [Gowal 2018] uses a loss function that includes two terms. The first term states a lower bound on the confidence on how well the expected output fits the class of the training example, and the second term states an upper bound on the confidence on how well the output fits a class that is not the expected output of the training example. The latter term applies to an entire neighborhood. Hence, this training makes sure that the trained ANN fits the training example and changing the input by some small amount does not produce an incorrect output.

Wong [Wong 2018] also considers an ANN used as a multi-class classifier. The paper [H] observes that, using a technique earlier presented by Ehlers [Ehlers 2017], the constraint that expresses ReLU activation function can be relaxed with three linear constraints (that form a triangle) if we know lower and upper bounds on the signal before the activation function is applied. Using this idea, Wong [Wong 2018] introduces an optimization problem, for a single given training example. For this training example, there is a correct class as output. Let us also select another class and call it the not-correct-class. We can now state an optimization problem: minimize the output (i.e., confidence level) of the neuron of the correct class minus the output of the neuron of the not-correct-class, subject to the constraint that the input to the ANN should be in a neighborhood of the input of the example. Since ReLU can be relaxed as mentioned above, it holds that this optimization problem is a linear program (LP). There are two things remaining however: (i) finding upper and lower bound, for each neuron, on the signal before activation function is applied, and (ii) finding some way to repeat the above for all neurons. The paper points out that one can obtain the dual problem to the original LP problem. This dual problem yields an upper bound on the optimal solution to the original LP problem. The paper proposes that we can simply select one feasible solution and plug that into the dual problem (rather than actually solving it). This yields a non-optimal solution for the dual problem. But it still provides a lower bound on the original problem. This has the advantage that it can be computed very quickly. The paper also uses this idea to compute lower and upper bounds on pre-activation signals. With these ideas, one can develop a robust training procedure. Instead of minimizing the loss function that expresses how the ANN fits the actual training examples, one minimizes a loss function that expresses how the ANN fits worst-case bounds of the training examples. The paper also points out that the bounds on performance expressed by the dual optimization problem can be thought of as a certificate of the performance of the ANN. This can be thought of as an explanation of the performance of the ANN (robustness).

Lin [Lin 2022] focuses on using correctness properties to drive training (similar to [Gowal 2018, Wong 2018]). The paper [Lin 2022] uses six ideas. First, it defines a measure (called concrete correctness loss function). For a given input, there is a set of correct output. This measure states how much an output from an ANN differs from the closest element in the set of correct output. If the output is

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

correct, then this distance is zero. Second, it introduces abstract domains and operations. For example, the number 2.5 may be an element in the concrete domain but the interval [2,5] may be an element in the abstract domain. A concrete element may be in an abstract domain. Operations can be formulated on abstract domains. For example, [2,5]+[11,18]=[13,23]. Third, the forward operation of an ANN (computing weighted sums and applying ReLU) can be formulated using the abstract domain. Fourth, training examples can be transformed to the abstract domain. Fifth, a training procedure can be formulated on the abstract domain (since both the ANN and the training examples are represented in the abstract domain). Sixth, the training procedure can detect which training example would benefit the most from a refinement of the abstraction. One could imagine (though not mentioned in the paper) that such an abstraction domain could also be used to explain the behavior of ANNs.

Raghunathan [Raghunathan 2018] focuses on using correctness properties to drive training. A key idea is to formulate the mathematical expression that yields the error bound so that although it depends on weight and biases in the ANN, it does not depend in the input. Hence, it applies to all inputs.

### 2.5.3 Provably correct ANN through training considering the environment

To have a correct ANN, it is necessary to (i) specify correctness, and (ii) make sure that the ANN is correct. We have already discussed techniques for the latter for the case that the ANN is a feedforward ANN (that is, no feedback). However, these works have three limitations. First, they assume an ANN in isolation (without its context). Second, they assume that correctness property is already given (i.e.., we do not need to figure it out). Third, they assume that the ANN does not rely on feedback. These limitations are serious for the case that an ANN is used to control an autonomous system (e.g., a robot) that operates in a physical environment.

Therefore, recent research [Sun 2019] has considered the problem of proving the correct behavior of an autonomous system that uses ANN. These works consider a state of the physical environment (e.g. x,y,z coordinates of the autonomous system, angles indicating heading, and time-derivatives of them) and it considers a state space. It defines a subset of states as safe states and another subset of states as unsafe states. An example of an unsafe state can be that the autonomous system hits an obstacle. These works aim to prove that the autonomous system never visits an unsafe state; for example, prove that the autonomous system never hits an obstacle.

It is assumed that the autonomous system has sensors and can compute a state based on these sensors and this state is fed as input to an ANN which computes an actuation command. This actuation command influences the physical environment. These works compute the transition function of the system; that is, the next state as a function of the current state. Based on this, one can compute the set of reachable states and compute the intersection between the set of reachable states and the set of unsafe. If this is empty, then the system is safe; otherwise, the system is unsafe. Computing this set of reachable states is very computationally demanding, however. Therefore, they use an overapproximation of the state space and define the transition function of that (because of the overapproximation, it becomes a transition relation). Computing the reachable states on this overapproximation is computationally cheaper. If this overapproximated on the reachable states does not intersect with the unsafe states, then the system is safe.

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

## 2.6 Current results at ERSA

ERSA is a recently started workshop focused on explainable verification. ERSA means explainability of real-time systems and their analysis. There have been two editions of ERSA so far. The first edition of ERSA presented the following papers/ideas/results:

1. Baruah [Baruah 2022] points out that in many schedulability analyses for real-time systems, there is a condition with existential quantification. This yields a witnesss. This witness can be thought of as an explanation.
2. Romagnoli [Romagnoli 2022] points out that in control theory, the Lyapunov function is used to prove stability of a feedback control system. This Lyapunov function can be thought of as an explanation.
3. Mitsch [Mitsch 2022] points out that for deductive proofs, if the architectural elements are used as symbols in the proof, then the proof is easier to understand.
4. Martins [Martins 2022] points out that a model checker can determine whether a certain property of a system is true; if it is false, then it can produce a counterexample, but it is hard for humans to understand the counterexample.

The second edition of ERSA presented the following papers/ideas/results:

1. Ahmad [Ahmad 2023] points out the use of simulation to highlight the output of schedulability analysis.
2. Romagnoli [Romagnoli 2023] builds on his previous result [Romagnoli 2022] to extend it for the case where it can happen that a controller fails to produce output (e.g., because of deadline misses).
3. Baruah [Baruah 2023] builds on his previous result[Baruah 2022] to also consider randomized verification.
4. Martins [Martins 2023] points out that LLMs can be used to explain properties of constraint satisfaction problems; specifically, the pigeon-hole principle.

## 2.7 Results at real-time systems conferences

There have been two papers in the area of real-time systems that deal with explainability, namely the following:

1. Maida [Maida 2022] presented a machine-checkable proof of a schedulability test and views this as an explanation.
2. Baruah [Baruah 2023b] builds on his previous work [Baruah 2022] and studies how to create small certificates.

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

### 2.8 Results at formal methods conferences

Cherukuri [Cherukuri 2022] presents a method to translate Linear Temporal Logic (LTL) formulas to natural language. The reason for creating this method is that most FM tools take a correctness specification as input and this correctness specification is often stated in LTL.

A blog post [Moy 2023] argues that it is important to start a new area called "Explainable Program Proofs." However, there is very little detail on how this should be achieved.

## 3 Situations where explainability is beneficial

Below, we list situations where explainability is beneficial. We start by listing situations that are in verification (our main focus), and then list situations that are outside verification (not in our main focus).

### 3.1 Explainability in verification

Figure 1 illustrates explainability in verification. It shows a formal methods tool (denoted by a filled black rectangle with the text "FM tool" in the figure). This formal methods tool takes as input a question and outputs an answer and an explanation to the answer. The question is whether a correctness property $\varphi$ holds for all executions that are possible in model M of system S.

The FM tools that are available in the research literature takes a question as input and produces an answer. Often it is as is shown in the figure. There are some variants, however. For example, some formal methods tool may output "undecided." Also, in this figure, the correctness property is "for all executions $\varphi$ holds." This can be rewritten as "for all executions it never happens that $\neg\varphi$ holds." This is called a safety property. The literature also includes liveness properties; they express that eventually something good happens. Liveness properties are relevant for systems where we do not have enough information to make sure that forward progress will happen but we still want to make sure that eventually something good happens. For example, consider a set of processes scheduled on a single processor where we do not know the scheduler and consider that we have a mutual exclusion protocol between them. In this case, we may be interested in proving that when a process requests the execution under mutual exclusion, then it will be granted eventually.

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

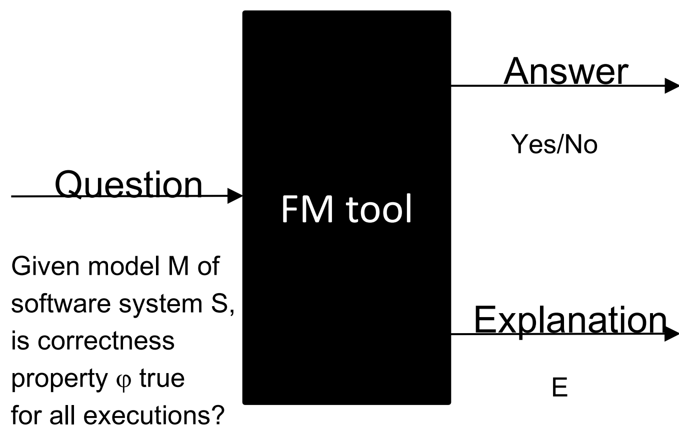


*Figure 1: Illustration of Explainability in Verification*

Figure 1 applies to a very large number of situations, including the following:

1. Real-time schedulability analysis. Here, M is the taskset and φ means meeting deadlines and all executions refers to all schedules that the system can generate.
2. Model checking of finite state machines. Here, M is the finite state machine and φ is a statement that no error state is reached.
3. Software model checking. Here, M is the source code of the computer program considered and φ is an assertion in the source code.
4. Control theory of state-feedback systems. Here, M is the differential equation describing the dynamics of the plant and how the actuation command is computed based on the state and φ is an assertion that the system is stable.
5. Buffer analysis in a computer networks. Here, M is the topology and the computer nodes in a computer network, including the queues in each computer node (for sending and transmitting) and queuing disciplines and traffic model. φ is an assertion that no buffer will overflow.
6. Mode confusion analysis. As an illustration, consider an aircraft and in the aircraft there are two computers and a human pilot. The aircraft can be in a mode; e.g., taxiing, or taking-off, flying horizontally, landing. A computer holds a view about what the current mode of the aircraft is; e.g., it is stored as a variable in a memory location in the aircraft. The human pilot holds a view about what the current mode of the aircraft is. Ideally, the views that the two computers have and the view that the human pilot has should be exactly the same. But if it is not, then there is mode confusion. A computer in an aircraft can be in a taxiing mode or take-off mode; the mode represents what the computer "believes" to be the state of the aircraft. Analogously, a human can be in taxiing mode or take-off mode; the mode represents what the human believes to be the state of the aircraft. Ideally, the mode that the human holds and the mode that the computer holds should be exactly the same. But in some cases they are not and

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

this is called Mode confusion. Mode confusion is a major cause of accidents but unfortunately, the research literature on FM tools to analyze mode confusion is currently scarce. Therefore, a mode confusion analysis can be thought of as taking as input M, which is a system of entities (humans or computers or both) and a description of how each of them switches modes, and also take as input φ, which is an assertion that at each time, for each pair of entities, the mode of these two entities do not differ too much (e.g., if they are in different modes, then these two modes are similar; or if they are in different modes, then one will switch to a new mode so that in a short amount of time, they will be in the same mode).

Figure 1 may also apply to worst-case execution time (WCET) analysis. In WCET analysis, one seeks an upper bound on the execution time of a given program. The execution time of a program depends not just on the source code or executable code of the program but also on the computer hardware it executes on. Therefore, WCET of a program depends on the hardware it executes on and hence WCET depends on information of the hardware. If M describes both the hardware and the computer program, then Figure 1. applies. However, in practice, one often does not know the hardware because hardware vendors do not disclose this information (or even if one has this information, it is very complex and often has errors). Therefore, in practice, WCET analysis must rely on measurements of the program or parts of the program. A WCET analysis may involve measurements and reasoning steps. Explainability can be applied nonetheless. For example, (i) explainability can be applied to the reasoning steps, and (ii) explainability can state which input was used for a test, and this can explain the output of a WCET analysis.

#### 3.1.1. Why was this output produced given input?

The research literature in AI has already presented a large number of methods for local explanation; for example LIME. These answer the question "Why was this output produced given this input?" It is tempting to believe that these can be applied also for explanation of formal verification.

There are number of reasons why LIME—and explanation methods like that in AI—do not work well for explainable verification; these include:

1. The output in explainable verification tends to be boolean; e.g., "yes/no" or "True/False" or in some cases "don't know." But the output for which LIME was designed is a scalar.
2. The output in explainable verification is about a predicate "forall executions x it holds that property φ is true." Hence, there is a special meaning of this Boolean output. When creating an explanation we want to take advantage of this special meaning, For example, in real-time schedulability analysis where the input is a taskset, the output is "forall schedules it holds that forall tasks for each job of this task the deadline of this job is met." In case the output is false, we can show a schedule and indicate a specific job of a specific task and show that its deadline is missed. Note that the objects that are used in the explanations here refer to the execution/schedule rather than the objects of the input (which is a taskset). Thus, in LIME, the explanation is an object that is of the same type as the input but in explainable verification, the explanation may be of a very different type than the input to the verification procedure. For example, in LIME, the explanation is a feature vector of input (which can be computed based

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

on the input) and the input and the explanation both refer to images. But in explainable verification, the input may be a taskset and the explanation may be a schedule and there is no single obvious way to compute the explanation (a schedule) from the taskset. A taskset describes a system but a schedule describes behavior.

3. In explainable verification, we are also interested in explaining the meaning of input and the meaning of the output. For example, in real-time scheduling, we may need to explain the meaning of $T_2$; we also may need to explain the meaning of the phrase "schedulable." The reason for this is that their meanings are different for different scheduling theories and hence they are also different for different tools that perform schedulability testing.

#### 3.1.2 Why is this condition true?

Many verification procedures are expressed as: if (condition is true), then (correctness property is true). Hence, we are often interested in evaluating this condition. And consequently, we are interested in explaining why this condition is true. In some cases, the above verification procedure is expressed as: if (∃y such that g(y) is true), then (correctness property is true). In this case, y is a certificate (also called witness). And then, we can use y as an explanation for the truth of this condition. Note that this only explains why the condition is true; it does not explain why the correctness property is true. To do that, we also need to explain why "if (condition is true), then (correctness property is true)" holds.

This type of explanation applies to many practically important situations; for example (i) in model checking the condition may be a set of reachable states, and (ii) in real-time scheduling theory, the condition may be a vector of response times of all tasks.

This type of explanation is discussed by Baruah [Baruah 2022] for real-time schedulability analysis.

#### 3.1.3 Why is this condition false?

Note that an explanation method for explaining why a condition is true does not necessarily work for explaining why a certain condition is false. A simple way to see this is as follows. Consider the condition: $\exists x\ x^2=9$. For this condition, x=3 is a witness that shows that this condition is true. Here, our explan3.1.3ation method is simply to present the witness. Let us now consider the condition $\exists x\ x=x+1$. Suppose that we want to explain why this condition is false. If we simply pick one value of x, then it can demonstrate that x=x+1 is false for this value of x. But it does not demonstrate that x=x+1 is false for all other values. One could subtract x on both sides and this yields "0=1" and this is simple enough that a human can see that it is false; but note that this is another type of reasoning than just giving a witness for an existential quantifier.

In formal languages and theoretical computer science, it is well-known that a method for generating a certificate for one problem/language is not necessarily useful for generating a certificate for the complement of the language. For example, it is not known whether NP=coNP. Most researchers suspect that NP≠coNP.

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

In the area of real-time schedulability analysis, it is noteworthy that a method for generating a certificate for schedulability typically cannot be used as a method for generating a certificate for unschedulability (and vice versa).

### 3.1.4 What does this input mean?

Our goal is to allow human users to trust FMs better. To do so, we need to eliminate doubts on trust. One such doubt is that the user may feed input to an FM tool and this input is a model of a system but this model is not what the user wants to feed as input. This can happen because the user does not understand the meaning of what is fed in. The following are examples of the need for explanation of the input:

1. In real-time schedulability analysis, the user may feed the input as a taskset where each task is described with parameters. For task 2, there may be a parameter $T_2$. However, for different tools and different theories, the parameter $T_2$ has different meanings. In some tools, $T_2$ means period of task 2 and the first time that task 2 arrives is at time 0. For some other tools, $T_2$ means period of task 2 and the first time that task 2 arrives is non-deterministic. Yet, for other tools, $T_2$ means minimum inter-arrival time of task 2.

2. In model checking of timed automata, it is common to introduce state invariants and transition guards. It is important for a user to understand whether a condition enables a transition or forces a transition. Also, clocks in timed automata bring the need to specify whether a location is a committed location or urgent location.

3. In software model checking, it is common to take source code (e.g., in the C programming language) as input. Often this source code is annotated with assertions and assumptions. A software model checker determines whether assertions are true; an example of this is the CBMC model checker for the C programming language. However, the software model checker needs to have a semantics of the programming language; in many cases this semantics is not specified and may be different from the semantics of the compiler used for this language. For example, we have experienced a case where an assertion in a C program is violated when running a compiled version of the C program but the CBMC model checker does not find it.

As can be seen from the above, it is helpful to provide explanation of the input to an FM tool so that the human user knows the meaning of what (s)he inputs to the tool.

### 3.1.5 What does this output mean?

FM tools typically produce an output; for example, the truth or falsity of an assertion. It is important to know what this truth and falsity refers to. For example, in real-time schedulability analysis, the user feeds the input as a taskset and it is assumed that the correctness property is schedulability. However, there is no explicit statement of this correctness property in the input; it is "hard coded." Nonetheless, the real-time schedulability analysis outputs an answer (False/True) based on this. Hence, it is important for the user to know what True/False refers to. In this case, it refers to schedulability. Also,

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

even if the human user knows that True/False refers to schedulability, it is important that the human user knows what schedulability means. In many theories, schedulability means that for all possible arrival times of jobs that are legal with respect to the model and for all possible execution times of jobs that are legal with respect to the model, it holds for the resulting schedule that for each task, for each job of this task, the deadline of the job is met. Note that this notion depends on how a schedule is generated at run-time and this depends on tie-breaking rules in the run-time scheduler (e.g., if two tasks have the same priority, then we need to apply a tie-breaking rule); typically tie-breaking rules are unspecified. For some models, the tie-breaking rule does not matter. But for some models (global-EDF scheduling on a multiprocessor), whether a taskset is schedulable depends on the tie-breaking rule used.

## 3.2 Explainability outside verification

There is a tension between the academic view of how to assure software systems versus the view that many industry practitioners (developers and certification authorities) hold. The former tends to hold the view that one should model a system and then mathematically prove its correctness (typically using FM tools). The previous section describes the role of explainability assuming this view. However, industry practitioners today tend to have another view which is based on (i) observing actual behavior in testing or operation, and (ii) requirements (like bi-directional tracing in DO-178C). For this reason, we discuss below explainability outside verification.

### 3.2.1 Why did this event occur in execution trace?

Unfortunately, a software developer often detects that the actual behavior of the software is different from the intended behavior. There is often an externally visible event that occurs but should not occur (or vice versa). In this case, the software developer often wants to explain this behavior. It often involves finding the root cause which could be either of the following:

1. Find the earliest computational step or state in the execution where there is a difference between actual behavior and intended behavior.
2. Find an element in a design artifact (e.g., executable code, source code, or architectural model) that causes this deviation.

A similar type of explanation is also needed in retro-active accident (or incident) investigation; e.g., an airplane has crashed and we want to know why. It is also needed in retro-active incident investigation.

### 3.2.2 What is the tracing between these two execution traces?

Software practitioners often mix development and assurance by starting with a high-level artifact and then making it more detailed. In some cases, the high-level artifact is executable and the more detailed artifact is also executable. And in this case, it is desirable to explain why two execution traces (one in the high-level artifact and one in the detailed artifact) do approximately the same thing. Often the more detailed artifact is exercised in a more realistic environment. Examples of such more realistic

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

environment include (i) hardware in-the-loop-simulation, (ii) real environment but that is not the one intended for operations, or (iii) real environment that is intended for operations. In this case, there is a need to explain tracing between two execution traces (one trace from high-level artifact and one trace from detailed artifact).

The academic literature offers an idea that compares two execution traces as well. The academic literature offers an idea called bisimulation which relates one execution trace in one system with an execution trace in another system.

#### 3.2.3 What is the tracing between a requirement on a high level and a set of requirements on a low level?

Many safety standards use requirements on different levels. For example, DO-178C specifies system requirements, high-level software requirements, low-level requirements, and source code (here four different levels). It is common to establish tracing between requirements on two different levels so that (i) the set of requirements on a lower level makes sure that a requirement on a higher level is satisfied, and (ii) for each requirement on a low level, there is a requirement on a high level to which the low-level requirement contributes. A similar mindset exists in the area of assurance case (where a claim plays the same role as a requirement).

In these situations, there is a need to form an explanation between a requirement on a high level and a set of requirements on a low level.

#### 3.2.4 Did the verification fail because of bad configuration?

For software systems with real-time requirements, we are often interested in proving (or disproving) correct timing of a set of threads/processes executing concurrently on a shared hardware platform. This depends on configuration; for example, if priority-based scheduling is used in the operating system, then it can happen that with Rate-Monotonic priority assignment, it is possible to prove correctness but with other priority assignment it is not. Therefore, verification can fail because of bad configuration.

In these situations, there is a need for a software tool to state that with the current configuration, it is not possible to prove the desired property but with another configuration it is. It is helpful to explain why different configurations makes a difference.

There is a variant of this where a system is underspecified. Consider for example a system where we do not know the priority assignment. In early design, it may be beneficial for a timing verification tool to output "I can't guarantee that timing will be correct; it depends on the priority assignment and you have not specified any priority assignment. Here is one priority assignment PR1 in which timing will be correct and here is another priority assignment PR2 in which timing will be incorrect. As you can see PR2 and PR1 differs in a very small way and this small difference yields a different in the verification output."

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

#### 3.2.5 Compiler warnings

There are many software tools that provide warnings on what appear to be a defect in an artifact. A common example of this is compilers. These often output a list of warnings when compiling source code of a computer program. Compilers can detect common programming errors (copy and paste errors, forgetting to initialize a variable) and recently some compilers also include FM tools that can prove properties of source code (using abstract interpretation). There are some signs of interest for compilers to provide explanations. A good example of this is gcc which today is not just capable of outputting warnings but is also capable of outputting explanations of warnings—see [FANALYZER 2024]. It can explain why a program does not halt and suggest that this non-halting is caused by a programming error. It can also visually explain buffer overflows.

# 4 Ideas for explainability

In this section, we present ideas for explanations in explainable verification—we neither focus on explanation of behavior nor focus on explanations in general. We can think of an explanation as a static object and an explanation as a process. Below, we focus mostly on the former but occasionally on the latter.

Recall from Figure 1. that we consider a question, an answer, and an explanation where:

1. The question is “given a model M of a system S, is correctness property true for all executions?,”
2. The answer is False/True/Undecided.

An explanation method for “yes” does not necessarily work as an explanation method for “no.” We discuss ideas for explanation methods below.

## 4.1 Explanation for the case that property is false

When the property is false, we can present a counterexample.

#### 4.1.1 Model checking: counterexample-of-execution-as-explanation

In model checking, if a property is false, then we can construct an execution in which the property is false. This can be shown to the human user. It has been reported, however, that counterexamples from model checkers are hard to read for humans [Martins 2022].

A possible idea would be to select a subset of states in the counterexample execution and present this to a human user. For example, if the counterexample is a sequence of states $\langle s_0,s_1,s_2,\ldots,s_{1000},s_{1001},s_{1002},\ldots,s_{2000}\rangle$ where $s_{2000}$ is the last state in the sequence and $s_{1000}$ is the first

state that violates a correctness property, then we can clearly remove all states after $s_{1000}$ and still have a counterexample. But we may decide to prune even further. We may find out that state $s_{200}$ is critical for understanding why the execution ended up in state $s_{1000}$ later on. Therefore, we may present the sequence of states $<s_1, s_{200}, s_{1000}>$ to the human user.

Another possibility is the following. We form an abstraction of the state space and then form the counterexample as a sequence of abstract states. Then, we convert each state in the counterexample to an abstract state. Now, we have a counterexample which is a sequence of abstract states. We can select a small subset of these and obtain a sequence of abstract states that explain why the property is false. There have been many previous works on abstracting a state space with the intention to speed up model checking. However, in this case, our purpose is different. Here, we would like to form an abstraction of the state space to facilitate human understanding. We would like to learn the human user's mental model and use that to drive the abstraction of states.

#### 4.1.2 Schedulability analysis: schedule-with-deadline-miss-as-explanation

In real-time schedulability analysis, if a taskset is unschedulable, then we can construct a legal schedule that the system can generate in which a deadline is missed. Similar to the above discussion on counterexample in model checking, we can select part of the schedule and present it to the user. Consider a schedule that leads to a deadline miss. Suppose that the first job that misses its deadline is the $q^{th}$ job of task $\tau_i$. One could present as a counterexample to the user the schedule from when the system starts until the deadline of the $q^{th}$ job of task $\tau_i$. But this may be a long schedule that is hard to understand. To create a simple schedule to present to the human user, we may want to select the schedule from the time of the arrival of the $q^{th}$ job of task $\tau_i$ until the deadline of the $q^{th}$ job of task $\tau_i$. We may want to make even further simplification of the counterexample. For example, there may be some tasks that have no influence on the execution of the $q^{th}$ job of task $\tau_i$ (these could be lower priority tasks). We may choose to not show those in the schedule.

We may simplify this further as follows. Replace execution of all other tasks than $\tau_i$ with a dummy task. This creates a schedule with only two tasks: $\tau_i$ and dummy task; here there is just a single job of task $\tau_i$. This can be presented to the human as an explanation for the claim that the taskset is unschedulable. When the human user has this simple explanation, (s)he can request more detailed explanations depending on her/her appetite for more details and depending on whether he/she trusts the simplest explanation.

There are additional ways to explain the counterexample. We may simply add up all the execution of the $q^{th}$ job of task $\tau_i$ and all the execution that prevents this execution (interference and blocking) and show that number to the user and also show the deadline ($D_i$). This also gives an explanation why a deadline was missed.

#### 4.1.3 WCET analysis: program-input-as-explanation

For WCET, we may want to explain why the execution time of a program P is greater than some bound B. This is interesting because it obviously implies that the WCET of a program P is greater than

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

B. One way to do so is to simply find the input to the program P that causes this long execution time, then give this input to the user. There may be a need to give additional information like initial state of the program.

#### 4.1.4 Software model checking: program-repair-as-explanation

A software model checker may find that an assertion in a source code file is false. To explain this, we may seek a repair for this program so that the assertion is true. Ideally, we would like the repair to be as small as possible (e.g., modifying as few lines of code as possible). There is a research area called program repair (also called automatic bug fixing) that does this. An example of such a tool is Wolverine [Wolverine 2023] which is based on LLM.

Note that here, we are not interested in actually repairing the program; we use a program repair simply to explain why a property is false.

### 4.2 Explanation for the case that property is true

When a property is true, we cannot simply present a counterexample to explain why it is true; we need to produce an argument that helps a human to get confidence in believing that the property is true.

#### 4.2.1 Model checking: proof-as-explanation

Some model checkers of finite state-machines today are able to produce a proof that the property is true. Let us assume (as is common) that a proof is a sequence of assertions where each assertion in the sequence is either (i) an axiom/assumption, or (ii) an assertion obtained from applying a proof rule on a set of assertions earlier in the sequence. Then, we can simply present this proof to the user and treat that as an explanation.

To make the explanation easier to understand for humans, we may prune this sequence so that it only contains the most important steps. This produces an explanation that may be easier for humans to understand. Yet another approach is to generate new proof rules so that with these proof rules, the sequence of assertions can be made much shorter.

Examples of work on software model checkers that produce proofs are: (i) developing proof harness and proof makefile for CBMC, and (ii) the use of Craig interpolation in software model checkers.

#### 4.2.2 Schedulability analysis: worst-case-situation-as-explanation

Real-time schedulability analyses often rely on a worst-case phasing. For example, in some task models, the Liu-and-Layland condition for critical instant applies. It states that the response time of a task is maximized for the case that its job arrives simultaneously with higher-priority tasks. In other models, when we are interested in whether task $\tau_i$ meets its deadline, we check all jobs of task $\tau_i$ in a so called busy window. In these cases, we may simply show a simulation of the job of this task of interest and show that for this particular phasing, the response time is at most the deadline. With this particular

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

phasing, we can make very small changes to the phasing and show that the response time of the task under interest (task $\tau_i$) is non-increasing. This gives an informal argument that this is the worst-case phasing (at least as a local maximum—not necessarily global maximum). We may want to do this as a process so that initially, we simply present the worst-case phasing and a Gantt chart of its schedule. Then, we state that this is the worst case. If the human user trusts that, then there is no need for further explanation. If the human user does not trust that, then the tool gives the aforementioned information to provide an explanation for why this is the worst case.

Some schedulability tests depends on a more complex form of phase. For example, in fixed-priority preemptive scheduling in time partitioned systems according to ARINC 653, the response time of a job of a task depends not only on its arrival relative to arrival times of other tasks but it also depends on the arrival relative to the start time of partition time windows.

Furthermore, some schedulability tests are sufficient but not exact; that is, they compute an upper bound on the response time but this computed number may be greater than the actual response time. In this case, one can create a schedule that is not necessarily legal schedule with respect to the taskset but nonetheless explains a response time. For example, considering a single job of task $\tau_i$, the schedulability analysis may have an over-approximation of carry-in execution of other tasks. Then, we can show the schedule from when this single job of task $\tau_i$ arrives until this single job finishes; at the time of arrival, we assume that other tasks have remaining execution that is equal to the assumed carry-in in the schedulability analysis.

#### 4.2.3 WCET analysis: program-input-as-explanation

Today, it is very hard to find the WCET of a program because the execution time of a program depends on hardware details (e.g., how does the memory controller work, how does the hardware prefetcher work, how does a cache miss influence the execution time considering an out-of-order processor). So in practice, we should not expect to be able to prove with 100% certainty a WCET of a program executing on a real processor (although on simple, non-realistic, hardware models this may work). However, given a number which we believe is the WCET and given an input that causes the program to execute with this believed WCET, we may want to argue that this is the WCET. Examples of such argument could be as follows.

Show that for each minor change in the input (flipping a bit), the execution time is non-increasing (hence, we have an argument that supports the assertion that the current input

1. is a local maximum of execution time). If the number of bits in the input is N, then this requires that we show N other inputs to the user. Typically, N is large so this type of explanation is not user-friendly. However, among these N other inputs, we may select a small number of them (e.g., 3 of them) so that they represent an important change in the program behavior (e.g., the execution takes another branch or a memory access that is a cache hit becomes a cache miss). This gives us a single input and certain perturbations and we can output an explanation: "Here is an input that we believe yields the WCET and here are perturbations to this input and these perturbation change the program behavior in important ways, yet they do not increase the execution time."

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

2. For this input, record the events that the execution of the program with this input experiences; in particular, identify the path in the control flow graph. Then, visualize this path and these events to the user. Now, the user should be allowed to modify events. For example, consider that part of the program is “if (cond) { exec_block1(); } else { exec_block2(); } Suppose that the path is such that cond is true and exec_block1 executes. Then, the user should be able to instruct the explanation tools that “I want to explore the case that cond is false.” The tool should then generate a new input so that in this place in the execution, cond is false and then the user should be able to run the target program with this input and see the execution time. In this way, the user can use his/her sources of doubts on whether the believed WCET is actually a WCET and use these doubts to check whether these doubts were well-founded. If the user has exercised the program sufficiently in this way, then eventually, the user’s doubts are eliminated. We can reason similarly with cache misses; e.g., one execution may have a cache hit at one location and the human users may state: “change it so that I get an input in which this memory access is a cache miss.”

#### 4.2.4 Software model checking: proof-as-explanation

This is similar to proof-as-explanation for model checking of finite state machines.

#### 4.2.5 Software model checking: worst-case-execution-as-explanation

Many assertions in software model checking is of the type: “assert(x<=B)” where x is a program variable and B is a constant. Note that in real-time systems, we often have requirements that for each execution it should hold that the response time is <= D. We can apply those explanation techniques also to software model checking if the assertion is expressed as above. In this way, we can obtain arguments that a certain execution causes x to be as large as possible. In a sense, this execution is the worst-case execution for the assertion “assert(x<=B)” We can visualize such an execution to the human user and this helps the user to trust the tool.

### 4.3 Explanation for either case

Here, we discuss explanations that may apply to the case that the property is false, the case that the property is true, or both.

#### 4.3.1 Set of all systems with true/false

Some systems are described by assigning values to parameters. For example, this is the case in real-time scheduling theory where we may describe a system with two tasks using the parameters $T_1$,$C_1$,$T_2$,$C_2$. In this case, we can view a system as a 4 dimensional vector. More generally, if a system has n tasks and each task is described by k parameters, then the description of the system is a n*k dimensional vector. We can compute the set of all vectors for which the answer is “True” and we can compute the set of all vectors for which the answer is “False.” We can then visualize these sets and

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

show these sets to the user. Since the user can see that the current taskset is in the set of schedulable tasksets, then the user can get more confidence in a schedulability analysis tool.

Humans are typically able to see visualizations of 2-dimensional vectors and 3-dimensional vectors. But the aforementioned vector spaces have much larger dimensionality. So we may need to apply dimensionality reduction techniques from exploratory data analysis.

#### 4.3.2 Dimensionality reduction of execution

We can represent an execution or schedule as a vector. We can apply dimensionality reduction of a vector so that it is easier to visualize. This idea can be combined with many of the aforementioned ideas.

#### 4.3.3 Structured argument

There are formats for structuring arguments. Examples of these include Goal Structuring Notation (GSN) and Claim Argument Evidence (CAE). These can be used to present explanations.

#### 4.3.4 LLM, chain-of-thought

Some of the explanations mentioned earlier may be difficult to understand. We can address this by simply taking an explanation and feeding it into an LLM and ask the LLM to create a simpler explanation. We can use prompting techniques to get better results; e.g., chain-of-thought prompting.

As an illustration, consider that we are interested in software model checking of source code file S and this source code file has been annotated with assumptions and an assertion/property and feeding S into a model checker (like CBMC) and the output from the model checker was that the property is false and it produced a counterexample CTX. Now, we want to obtain a human-understandable explanation E for this counterexample. We can do it as follows. We generate another source code file S' for which the property is also false and it produces a counterexample CTX' and we as human decide that E' is a good explanation for this. Then, we can feed the following to the LLM: Given S', CTX', E', S, CTX, find E. That is, S', CTX', E' helps the LLM to understand the general idea about what we mean with explanation. Then S and CTX tells the LLM the specific situation for which we want the explanation.

#### 4.3.5 LLM, interactive book

Tyler Cowen has recently [Cowen 2023] written a book "Greatest Economist of All Times." This is not a book with pages; instead, it is a published LLM that allows a human reader to query it and then it responds. In this case, the content of a book is conveyed by allowing users to query the content. This has the advantage that the user's curiosity can drive the interaction and that the user can get content that (s)he is interested in.

One could potentially transfer this idea to explainable verification. Specifically, when an FM tool has produced an output (False/True) and exposed some internal information, then we can think of this internal information as content, and we train an LLM based on this content. Then, this LLM is presented

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

to the human user. Note that this involves re-training the LLM each time the human user uses an FM tool. One may think this is very expensive. However, there are techniques for incrementally training/fine tuning LLMs—see for example LoRA [Hu 2023].

#### 4.3.6 Generative AI, video generation

Recent advances in generative AI have provided the capability to generate video based on a string of text that specifies the content of the video. An example of this is SORA from OpenAI. This could be useful for creating explanations. For example, a formal verification tool that has found a counterexample could generate an animation of state transitions that generate a counterexample. If this counterexample can only occur in a certain context, then this can be part of the animation as well. For example, if the input to the computer program that triggers this counterexample can only occur in an aircraft when it lands, then the video may show an aircraft that lands in addition to the visualization of the state changes of the software.

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

## References

**[AFMAN 2020]**
AFMAN 91-119 Safety Design Criteria For Nuclear Weapon Systems Software, March 2020.

**[Ahmad 2023]**
Tanveer Ali Ahmad, Muhammad; Pestana, Jesus; Batista, Leandro; & Baunach, Marcel. Budget-Based Explainable Schedulability Analysis for Automotive Applications. In *2nd International Workshop on Explainable Real-time Systems and their Analysis (ERSA 2023).* December 2023. https://sites.google.com/view/ersa23

**[Angluin 1987]**
Angluin, Dana. Learning Regular Sets from Queries and Counterexamples. *Information and Computation*. Vol 75. Issue 2. November 1987. Pages 87–106. https://www.sciencedirect.com/science/article/pii/0890540187900526

**[Ball 2004]**
Ball, Thomas; Cook, Byron; Levin, Vladimir; & Rajamani, Sriram K. SLAM and Static Driver Verification: Technology Transfer of Formal Methods inside Microsoft. Technical Report, MSR-TR-2004-08. 2004. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2004-08.pdf

**[Ball 2011]**
Ball, Thomas; Levin Vladimir; & Rajamani, Sriram K. A Decade of Software Model Checking with SLAM. *Communications of the ACM*. Vol 54. No 7. July 2011. Pages 68–76. https://dl.acm.org/doi/abs/10.1145/1965724.1965743

**[Bach 2015]**
Bach, Sebastian; Binder, Alexander; Montavon, Grégoire; Klauschen, Frederick; Müller, Klaus-Robert; & Samek, Wojciech. On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *PLoS ONE*. Vol 10. Issue 7. July 2015. https://journals.plos.org/plosone/article/file?id=10.1371/journal.pone.0130140&type=printable

**[Baruah 2022]**
Baruah, Sanjoy & Ekberg Pontus. Certificates of Real-Time Schedulability. In *1st International Workshop on Explainable Real-time Systems and their Analysis (ERSA 2022).* December 2022. https://sites.google.com/view/ersa22

**[Baruah 2023]**
Baruah, Sanjoy & Ekberg Pontus. Efficient Explainability of Real-Time Schedulability. In *2nd International Workshop on Explainable Real-time Systems and their Analysis (ERSA 2023).* December 2023. https://sites.google.com/view/ersa23

**[Baruah 2023b]**
Baruah, Sanjoy & Ekberg Pontus. Towards Efficient Explainability of Schedulability Properties in Real-Time Systems. In *35th Euromicro Conference on Real-Time Systems (ECRTS 2023).* July 2023. https://drops.dagstuhl.de/storage/00lipics/lipics-vol262-ecrts2023/LIPIcs.ECRTS.2023.2/LIPIcs.ECRTS.2023.2.pdf

**[Bermúdez 2020]**
Bermúdez, José Luis. Cognitive Science: An Introduction to the Science of the Mind. Cambridge University Press. 3rd Edition, 2020. https://www.cambridge.org/highereducation/books/cognitive-science/8399DB4BA11A1B7F6AABC104FD3E6323#overview

**[Bertsimas 2017]**
Bertsimas, Dimitris & Dunn, Jack. Optimal classification trees. In *Machine Learning*. Vol 106. Number 7. Pages 1039-1082. 2017. https://link.springer.com/content/pdf/10.1007/s10994-017-5633-9.pdf

**[Biere 1999]**
Biere, Armin; Cimatti, Alessandro; Clarke, Edmund; & Zhu, Yunshan. Symbolic Model Checking without BDDs. Pages 193-207. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1999).* January 1999. https://link.springer.com/chapter/10.1007/3-540-49059-0_14

**[Chechik 2005]**
Chechik, Marsha & Gurfinkel, Arie. A Framework for Counterexample Generation and Exploration. Pages 220-236. In *Fundamental Approaches to Software Engineering (FASE 2005).* April 2005. https://link.springer.com/content/pdf/10.1007/978-3-540-31984-9_17.pdf

**[Cherukuri 2022]**
Cherukuri, Himaja; Ferrari, Alessio; & Spoletini, Paola. Towards Explainable Formal Methods: From LTL to Natural Language with Neural Machine Translation. Pages 79-86. In *Requirements Engineering: Foundation for Software Quality: 28th International Working Conference (REFSQ 2022).* March 2022. https://link.springer.com/chapter/10.1007/978-3-030-98464-9_7

**[Clarke 2018]**
Jhala Ranjit; Podelski Andreas; & Rybalchenko Andrey. Predicate Abstraction for Program Verification. In *Handbook of Model Checking*. Clarke, Edmund M.; Henzinger, Thomas A.; Veith Helmut; & Bloem Roderick. [editors]. Springer. Pages 447-491. 2018. https://link.springer.com/content/pdf/10.1007/978-3-319-10575-8_15?pdf=chapter%20toc

**[Cowen 2023]**
Cowen, Tyler. *Greatest Economist of All Times*. https://goatgreatesteconomistofalltime.ai/en

**[Craik 1943]**
Craik, Kenneth. *The Nature of Explanation*. Cambridge University Press. 1943. https://www.amazon.com/Nature-Explanation-Kenneth-K-Craik/dp/0521094453

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

**[DARPA 2022]**
DARPA Explainable Artificial Intelligence (XAI) Program. 2022. https://www.darpa.mil/program/explainable-artificial-intelligence.

**[Davis 2007]**
Davis, Robert I.; Burns Alan; Brill, Reinder J.; & Lukkien, Johan J. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-time Systems*. Vol 35. 2007. Pages 239–272. https://link.springer.com/content/pdf/10.1007/s11241-007-9012-7.pdf

**[Deutsch 1997]**
Deutsch, David. *The Fabric of Reality*. Penguin Random House. ISBN 9780140275414. 1998. https://www.penguinrandomhouse.com/books/330123/the-fabric-of-reality-by-david-deutsch/

**[Deutsch 2009]**
Deutsch, David Deutsch. *A new way to explain explanation*. TEDGlobal, 2009. https://www.ted.com/talks/david_deutsch_a_new_way_to_explain_explanation?language=en

**[Deutsch 2012]**
Deutsch, David. The beginning of Infinity—Explanations that Transform the World. Penguin Book, 2012. https://www.thebeginningofinfinity.com/

**[Deutsch 2023]**
The Deutsch Files II, 2003, 58:20. Youtube. https://www.youtube.com/watch?v=I6GNK6BR4E8

**[Diemert 2023]**
Diemert, Simon; Goodenough, John; Joyce, Jeff; & Weinstock, Charles B. Incremental Assurance through Eliminative Argumentation. *Journal of System Safety.* Volume 58. Number 1. March 2023. https://jsystemsafety.com/index.php/jss/article/view/215

**[DO-178C 2011]**
*Software Considerations in Airborne Systems and Equipment Certification*, RTCA, DO-178C, 2011.

**[Dvijotham 2018]**
(Dj) Dvijotham, Krishnamurthy; Stanforth, Robert; Gowal, Sven; Mann, Timothy; & Kohli, Pushmeet. A Dual Approach to Scalable Verification of Deep Networks. Pages 550–559. In *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence (UAI),* August 2018. https://auai.org/uai2018/proceedings/papers/204.pdf

**[Ehlers 2017]**
Ehlers, Rüdiger. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *Automated Technology.* May 2017. https://www.semanticscholar.org/paper/Formal-Verification-of-Piece-Wise-Linear-Neural-Ehlers/333416708c80d0c163ca275d1b190b1f2576fa5f

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

**[Ericson 2016]**
Ericson II, Clifton A. *Hazard Analysis Techniques for System Safety*. Wiley. ISBN: 978-1-119-10168-0. Second Edition, 2016. https://www.wiley.com/en-sg/Hazard+Analysis+Techniques+for+System+Safety%2C+2nd+Edition-p-9781119101680

**[ERSA 2022]**
1st International Workshop on Explainable Real-time Systems and their Analysis (ERSA 2022). https://sites.google.com/view/ersa22

**[ERSA 2023]**
2nd International Workshop on Explainable Real-time Systems and their Analysis (ERSA 2023). https://sites.google.com/view/ersa23

**[Falco 2021]**
Falco, Gregory; Shneiderman, Ben; Badger, Julia; Carrier, Ryan; Dahbura, Anton; Danks, David; Eling, Martin; Goodloe, Alwyn; Gupta, Jerry; Hart, Christopher; Jirotka, Marina; Johnson, Henric; LaPointe, Cara; Llorens, Ashley J.; Mackworth, Alan K.; Maple, Carsten; Pálsson, Sigurður Emil; Pasquale, Frank; Winfield, Alan; & Yeong, Zee Kin. Governing AI safety through independent audits. *Nature Machine Intelligence*. Vol 3. 2021. Pages 566–571. https://www.nature.com/articles/s42256-021-00370-7

**[FANALYZER 2024]**
Fanalyzer switch for gcc compiler. https://developers.redhat.com/articles/2024/04/03/improvements-static-analysis-gcc-14-compiler#

**[Friedman 1974]**
Fridman, Jerome H. & Tukey, John W. A projection Pursuit Algorithm for Exploratory Data Analysis. *IEEE Transactions on Computers*. Vol 9. No 9. 1974. Pages 881–890. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1672644

**[Garavel 2020]**
Garavel, Hubert; Beek, Maurice H.ter; & Pol Jacovande. The 2020 Expert Survey on Formal Methods. Pages 3-69. In *International Conference on Formal Methods for Industrial Critical Systems (FMICS 2020),* August 2020. https://link.springer.com/content/pdf/10.1007/978-3-030-58298-2_1.pdf?pdf=inline%20link

**[Goodfellow 2015]**
Goodfellow, Ian J.; Shlens, Jonathon; & Szegedy, Christian. Explaining and Harnessing Adversarial Examples. Pages 3-69. In *3rd International Conference on Learning Representations (ICLR 2015),* May 2015. https://arxiv.org/abs/1412.6572

**[Gowal 2018]**
Gowal, Sven; (Dj) Dvijotham, Krishnamurthy; Stanforth, Robert; Bunel, Rudy; Qin, Chongli; Uesato, Jonathan; Arandjelovic, Relja; Mann, Timothy; & Kohli, Pushmeet. On the Effectiveness of Interval

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

Bound Propagation for Training Verifiably Robust Models. Pages 3-69. In *NeurIPS SECML 2018 Workshop,* December 2018. https://arxiv.org/abs/1810.12715

**[Graf 1997]**
Graf, Susanne & Saïdi, Hassen. Construction of Abstract State Graphs with PVS. Pages 72-83. In *9th International Conference on Computer Aided Verification (CAV 1997),* June 1997. https://link.springer.com/chapter/10.1007/3-540-63166-6_10

**[Gupta 2021]**
Gupta, Sharmi Dev; Genc Begum; & O'Sullivan, Barry. Explanation in Constraint Satisfaction: A Survey. Pages 4400-4407. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence Survey Track*. August. 2021. https://www.ijcai.org/proceedings/2021/601

**[Hagras 2018]**
Hagras, Hani. Toward Human-Understandable Explainable AI. *IEEE Computer Magazine*. Vol 51. Issue 9. September 2018. Pages 28–36. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&ar-number=8481251

**[Hilderman 2021]**
Hilderman, Vance. *The Aviation Development EcoSystem*. Aviation Press International, 2021. https://afuzion.com/books-from-afuzion-aviation-development/

**[Hinman 2005]**
Hinman, Peter G. *Fundamental of Mathematical Logic*. Taylor & Francis, 2005. https://www.taylor-francis.com/books/mono/10.1201/b10690/fundamentals-mathematical-logic-peter-hinman

**[Hoffman 2018]**
Hoffman, Robert R.; Mueller, Shane T.; Klein Gary; Litman Jordan. Metrics for Explainable AI: Challenges and Prospects. 2018. https://arxiv.org/ftp/arxiv/papers/1812/1812.04608.pdf

**[Holzinger 2018]**
Holzinger , Andreas. From Machine Learning to Explainable AI. In *Proceedings of the 2018 World Symposium on Digital Intelligence for Systems and Machines (DISA 2018)*. August. 2018. https://iee-explore.ieee.org/document/8490530

**[Hu 2023]**
Hu. Edward J.; Shen. Yelong; Wallis, Phillip; Allen-Zhu, Zeyuan; Li Yuanzhi; Wang Shean; Wang Lu; & Chen Weizhu. LoRA: Low-Rank Adaptation of Large Language Models. https://arxiv.org/pdf/2106.09685.pdf

**[ISO26262part8 2018]**
ISO 26262 Road vehicles — Functional safety — Part 8: Supporting processes.

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

**[Kaleeswaran 2023]**
Kaleeswaran Arut Prakash. Explanation of the Model Checker Verification Results. PhD thesis. Humboldt-University in Berlin. December 2023. https://edoc.hu-berlin.de/bitstream/handle/18452/28605/dissertation_kaleeswaran_arut_prakash.pdf

**[Katz 2017]**
Katz, Guy; Barrett, Clark; Dill, David; Julian, Kyle; & Kochenderfer, Mykel. Reluplex: An efficient SMT solver for verifying deep neural networks. Pages 97–117. In *29th International Conference on Computer Aided Verification (CAV 2017)*. Heidelberg. Germany. July 24–28. 2017. https://arxiv.org/abs/1702.01135

**[Katz 2019]**
Katz, Guy; Huang, Derek A.; Ibeling, Duligur; Julian, Kyle; Lazarus, Christopher; Lim, Rachel; Shah, Parth; Thakoor, Shantanu; Wu, Haoze; Zeljić, Aleksandar; Dill, David; Kochenderfer, Mykel J.; & Barrett, Clark. The marabou framework for verification and analysis of deep neural networks. Pages 443–452. In *31st International Conference on Computer Aided Verification (CAV 2019)*. New York City, NY, USA. July 15–18. 2019. https://theory.stanford.edu/~barrett/pubs/KHI+19.pdf

**[Katz 2022]**
Katz, Guy; Barrett, Clark; Dill, David; Julian, Kyle; & Kochenderfer, Mykel. Reluplex: a calculus for reasoning about deep neural networks. *Formal Methods in System Design*. Vol 60. 2022. Pages 87–116. https://link.springer.com/article/10.1007/s10703-021-00363-7

**[Keil 2006]**
Keil, Frank C. Explanation and Understanding. *Annual Review of Psychology*. Vol 57. 2006. Pages 227–254. https://www.annualreviews.org/doi/abs/10.1146/annurev.psych.57.102904.190100

**[Kelly 1998]**
Kelly, Timothy Patrick. *Argument Safety—A Systematic Approach to Managing Safety Cases*. PhD thesis, 1998. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=81d2e41a5673a8d4a0d7c78ca3d0b0ff26165991

**[Krug 2010]**
Krug. Steve. *Rocket Surgery Made Easy*. New Riders, 2010. https://sensible.com/rocket-surgery-made-easy/

**[Krug 2014]**
Krug, Steve Krug. Don't make me think. 3rd Edition, New Riders, 2014. https://sensible.com/dont-make-me-think/

**[Linn 2022]**
Lin, Xuankang; Zhu, He; Samanta, Roopsha; & Jagannathan, Suresh. ART: Abstraction Refinement-Guided Training for Provably Correct Neural Networks. Pages 148–157. In *Proceedings of the 20th Conference on Formal Methods in Computer-Aided Design (FMCAD 2020)*. September 21–24. 2020. https://arxiv.org/abs/1907.10662

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

**[Liu 2023]**
Liu, Ziming; Gan, Eric; & Tegmark, Max. Seeing is believing: Brain-inspired modular training for mechanistic interpretability. 2023. https://arxiv.org/pdf/2305.08746.pdf

**[Lipton 2001]**
Lipton, Peter. What makes an explanation. In *Explanation: Theoretical Approaches and Applications*. Hon, Giora & Rakover, Sam S. [editors]. Springer. Pages 43-59. 2001. https://link.springer.com/content/pdf/10.1007/978-94-015-9731-9_2.pdf

**[Lundberg 2017]**
Lundberg, Scott M. & Lee, Su-In.. A unified approach to interpreting model predictions. Pages 4768–4777. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17).* Long Beach, California USA. December 4-9, 2017. https://dl.acm.org/doi/10.5555/3295222.3295230

**[Maduma 2019]**
Maduma, Prashan; Miller, Tim; Sonenberg, Liz; & Vetere, Frank. Explainable Reinforcement Learning Through a Causal Lens. Pages 2493-2500. In *Association for the Advancement of Artificial Intelligence (AIAA-20).* New York City, NY, USA. February 7−12. 2020. https://arxiv.org/abs/1905.10958

**[Maida 2022]**
Maida, Marco; Bozhko, Sergey; & Brandenburg; Björn B. Foundational Response-Time Analysis as Explainable Evidence of Timeliness. Pages 1–25. In *34th Euromicro Conference on Real-Time Systems (ECRTS 2022).* Modena, Italy. July 5-8, 2022. https://drops.dagstuhl.de/storage/00lipics/lipics-vol231-ecrts2022/LIPIcs.ECRTS.2022.19/LIPIcs.ECRTS.2022.19.pdf

**[Mansur 2020]**
Mansur, Muhammad Numair; Christakis, Maria; Wüstholz, Valentin; & Zhang, Fuyuan. Detecting critical bugs in SMT solvers using blackbox mutational fuzzing. Pages 701–712. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020).* November 8-13, 2020. https://dl.acm.org/doi/pdf/10.1145/3368089.3409763

**[Martins 2022]**
Martins, Ruben. Towards Explainable Formal Verification. In *1st International Workshop on Explainable Real-time Systems and their Analysis (ERSA 2022).* December 2022. https://sites.google.com/view/ersa22

**[Martins 2023]**
Martins, Ruben. Transforming Logic into Language: Bridging the Gap with Large Language Models. In *2nd International Workshop on Explainable Real-time Systems and their Analysis (ERSA 2023).* December 2023. https://sites.google.com/view/ersa23

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

**[Michaud 2024]**
Eric J. Michaud, Isaac Liao, Vedang Lad, Ziming Liu, Anish Mudide, Chloe Loughridge, Zifan Carl Guo, Tara Rezaei Kheirkhah, Mateja Vukelić, Max Tegmark. Opening the AI black box: Program Synthesis via Mechanistic Interpretability. https://arxiv.org/abs/2402.05110

**[Mellon 2023]**
Mellon, Jeffrey & Worrell, Clarence. Explainability in Cybersecurity Data Science. In *SEI blog post series.* November 2023. https://insights.sei.cmu.edu/blog/explainability-in-cybersecurity-data-science/

**[Mitsch 2022]**
Mitsch, Stefan. Formal Artifacts as Explanations for System Correctness in Cyber-Physical Systems. In *1st International Workshop on Explainable Real-time Systems and their Analysis (ERSA 2022).* December 2022. https://sites.google.com/view/ersa22

**[Moy 2023]**
Moy, Yannick. *Explainable Program Proofs*. April 4, 2023. https://blog.adacore.com/explainable-program-proofs

**[Norman 2013]**
Norman, Don. The design of everyday things. Basic Books. 2013. https://www.amazon.com/Design-Everyday-Things-Revised-Expanded/dp/0465050654

**[Park 2021]**
Park, Jiwon; Winterer, Dominik; Zhang, Chengyu; & Su Zhendong. Generative type-aware mutation for testing SMT solvers. Pages 1–19. In *Proceedings of the ACM on Programming Languages (OOPSLAA).* October, 2021. https://dl.acm.org/doi/pdf/10.1145/3485529

**[Pearl 2019]**
Pearl, Judea. The Seven Tools of Causal Inference, with Reflections on Machine Learning. *Communications of the ACM*. Vol 62. Issue 3. March 2019. Pages 54–60. https://dl.acm.org/doi/pdf/10.1145/3241036

**[Raghunathan 2018]**
Raghunathan, Aditi; Steinhardt, Jacob; & Liang, Percy. Certified Defenses against Adversarial Examples. In *Proceedings of the Sixth International Conference on Learning Representations (ICLR).* May, 2018. https://arxiv.org/pdf/1801.09344.pdf

**[Ramaprasaath 2017]**
Selvaraju, Ramprasaath R.; Cogswell, Michael; Das, Abhishek; Vedantam, Ramakrishna; Parikh, Devi; & Batra, Dhruv. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. Pages 336–359. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV).* Venice, Italy. October, 2017. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8237336

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

**[Ribeiro 2016]**
Ribeiro, Marco Tulio; Singh, Sameer; & Guestrin, Carlos. "Why Should I Trust You?" Explaining the Predictions of Any Classifier. Pages 336–359. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD).* San Francisco, CA, USA. August, 2016. https://www.kdd.org/kdd2016/papers/files/rfp0573-ribeiroA.pdf

**[Romagnoli 2022]**
Romagnoli, Raffaele. Understanding Safety of Linear Real-Time Systems from Lyapunov Theory and Quadratic Boundedness. In *1st International Workshop on Explainable Real-time Systems and their Analysis (ERSA 2022).* December 2022. https://sites.google.com/view/ersa22

**[Romagnoli 2023]**
Romagnoli, Raffaele. Explaining Quadratic Boundedness for Latency Mitigation and Safety Assurance in Edge-Cloud Computing. In *2nd International Workshop on Explainable Real-time Systems and their Analysis (ERSA 2023).* December 2023. https://sites.google.com/view/ersa23

**[Rosenthal 2016]**
Stephanie Rosenthal, "Why did the Robot do That?," November 2016. Available at: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=474300.

**[SEIGreybook 2022]**
Pages 87-96 in SEI Grey book. https://wiki-int.sei.cmu.edu/confluence/download/attachments/446335837/02-FY23-24-CMU-SEI-GB-JAC-EG%20%283%29.pdf?version=2&modificationDate=1659038941357&api=v2

**[Shrikuma 2017]**
Shrikuma, Avanti; Greenside, Peyton; & Kundaje, Anshul. Learning Important Features Through Propagating Activation Differences. Pages 3145–3153. In *Proceedings of the 34th International Conference on Machine Learning (ICML).* Sydney, Australia. August, 2017. https://arxiv.org/abs/1704.02685

**[Sun 2019]**
Sun, Xiaowu; Khedr, Haitham; & Shoukry, Yasser. Formal verification of neural network controlled autonomous systems. Pages 147–156. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC).* Montreal, Quebec, Canada. April, 2019. https://dl.acm.org/doi/pdf/10.1145/3302504.3311802

**[Szegedy 2013]**
Szegedy, Christian; Zaremba, Wojciech; Sutskever, Ilya; Bruna, Joan; Erhan, Dumitru; Goodfellow, Ian &; Fergus, Ro. Intriguing properties of neural networks. In *Proceedings of the 2nd International Conference on Learning Representations.* Banff, AB, Canada. August, 2014. https://arxiv.org/abs/1312.6199

**[TDB 2024]**
Transformer Debugger (TDB). https://github.com/openai/transformer-debugger

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

**[Wikipedia 2023a]**
Scientific Explanation at Stanford Encyclopedia of Philosophy. https://plato.stanford.edu/entries/scientific-explanation/

**[Wikipedia 2023b]**
Metaphysical Explanation at Stanford Encyclopedia of Philosophy.https://plato.stanford.edu/entries/metaphysical-explanation/

**[Wikipedia 2023c]**
Explanation in Mathematics at Stanford Encyclopedia of Philosophy. https://plato.stanford.edu/entries/mathematics-explanation/

**[Wikipedia 2023d]**
Modified condition/decision coverage. https://en.wikipedia.org/wiki/Modified_condition/decision_coverage

**[Wikipedia 2024]**
Principle of sufficient reason. https://en.wikipedia.org/wiki/Principle_of_sufficient_reason

**[Winterer 2020]**
Winterer, Dominik; Zhang, Chengyu; & Su, Zhendong. On the unusual effectiveness of type-aware operator mutations for testing SMT solvers. Pages 1-25. In *Proceedings of the ACM on Programming Languages (OOPSLA).* Chicago, Illinois, USA. November 2020.
https://dl.acm.org/doi/pdf/10.1145/3428261#:~:text=We%20propose%20type%2Daware%20operator,test%20cases%20for%20SMT%20solvers

**[Wolverine 2023]**
https://github.com/biobootloader/wolverine

**[Wong 2018]**
Wong, Eric & Kolter, J. Zico. Provable defenses against adversarial examples via the convex outer adversarial polytope. Pages 5283-5292. In *Proceedings of the Thirty-fifth International Conference on Machine Learning (ICML).* Stockholm. Sweden. June 2018. https://arxiv.org/abs/1711.00851

**[Zhang 2019]**
Zhang, Chengyu; Su, Ting; Yan, Yichen; Zhang, Fuyuan; Pu, Geguang; Zhendong Su. Finding and understanding bugs in software model checkers. Pages 763–773. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019).* Talinn, Estonia. August 2019.
https://dl.acm.org/doi/pdf/10.1145/3338906.3338932

DISTRIBUTION STATEMENT A: This material has been approved for public release and unlimited distribution.

**[Zhao 2023]**
Zhao Haiyan; Chen, Hanjie; Yang, Fan; Liu, Ninghao; Deng, Huiqi; Cai, Hengyi; Wang, Shuaiqiang; Yin, Dawei; & Du, Mengnan. Explainability for Large Language Models: A Survey. 2023. https://arxiv.org/pdf/2309.01029.pdf

**[Zhou 2016]**
Zhou, Bolei; Khosla, Aditya; Lapedriza, Agata; Oliva, Aude; & Torralba, Antonio. Learning Deep Features for Discriminative Localization. Pages 2921-2929. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* Las Vegas, NV, USA. June 2016. https://arxiv.org/pdf/1512.04150.pdf

## Legal Markings

Copyright 2024 Carnegie Mellon University.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License. Requests for permission for non-licensed uses should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM24-0413

## Contact Us

Software Engineering Institute
4500 Fifth Avenue, Pittsburgh, PA 15213-2612

**Phone**: 412/268.5800 | 888.201.4479
**Web**: www.sei.cmu.edu
**Email**: info@sei.cmu.edu