PSL UNIVERSITÉ PARIS

HABILITATION À DIRIGER DES RECHERCHES

DE L'UNIVERSITÉ PSL

Présentée à l'École normale supérieure

Analyses statiques pour les propriétés, les programmes et le public de demain

Static Analyses for the Properties, Programs, and People of Tomorrow

Présentation des travaux par

Caterina URBAN

Le 30 septembre 2025

Discipline

INFORMATIQUE

Composition du jury :

Sandrine, BLAZY Université de Rennes, FR	*Présidente*
Matthieu MARTEL Université de Perpignan, FR	*Rapporteur*
David NAUMANN Stevens Institute of Technology, US	*Rapporteur*
Corina PASAREANU Carnegie Mellon University, US	*Rapporteur*
Patrick COUSOT New York University, US	*Examinateur*
Laure GONNORD Grenoble INP - UGA, FR	*Examinatrice*
João MARQUES-SILVA Université de Lleida, ES	*Examinateur*
Antoine MINÉ Sorbonne Université, FR	*Examinateur*
Xavier RIVAL Inria & École Normale Supérieure, FR	*Examinateur*
Jan VITEK Northeastern University, US	*Examinateur*

ENS | PSL

Contents

1

Career and Work

Parcours et travaux

I am interested in developing methods and tools to enhance the quality and reliability of computer software and, more broadly, to help understanding complex software systems. My main area of expertise is *static analysis*, particularly within the framework of *abstract interpretation*, a unifying mathematical theory for describing and comparing the behavior of computer programs at different levels of abstraction. However, in my research activity, I have also gained expertise in *software model checking* and *deductive program verification*. I thus have a broad knowledge that spans the whole spectrum of formal methods.

Software defects can have catastrophic consequences in safety-critical areas such as transportation, nuclear power generation, and medical systems. Moreover, as our reliance on software systems grows, we become increasingly vulnerable to software defects in our daily lives. This risk is further amplified nowadays by the rise of machine learning-based software, which plays an increasingly important role in high-stakes decision making, despite its often opaque and unpredictable behavior. Ensuring the robustness, reliability, and explainability of such systems is an urgent challenge, and has been a central focus of my research since 2018.

1.1 Pushing for More Advanced Static Analyses

Proving that computer programs behave correctly is a difficult problem because it is mathematically *undecidable*: it is impossible to construct an algorithm that always gives a precise yes-or-no answer regarding the correctness of a program. Static program analysis offers a solution to this problem by relaxing the requirement of returning a yes-or-no answer: the reasoning is based on an automatically computed approximation of the behavior of the given program, which results in a positive *yes* answer (if all computed executions satisfy the desired correctness property, cf. Figure 1.1) or an inconclusive *unknown* answer (otherwise, cf. Figure 1.2). In the latter case, the analysis returns a warning, which either corresponds to a real error in the program or is a false positive due to the approximation introduced by the analysis (e.g., the red region in Figure 1.2). The main challenge is then to develop static analyses that are precise enough to return an inconclusive answer as seldom as possible.

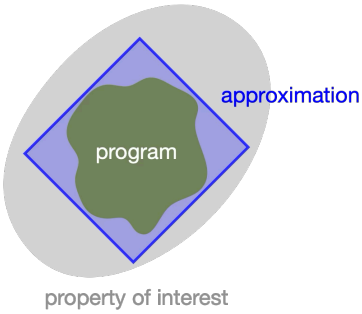


Figure 1.1: Sound approximation.

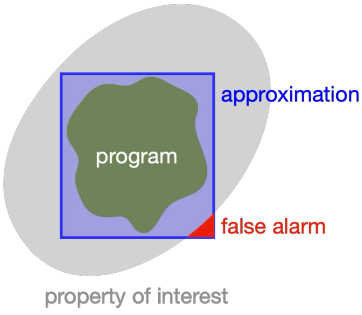
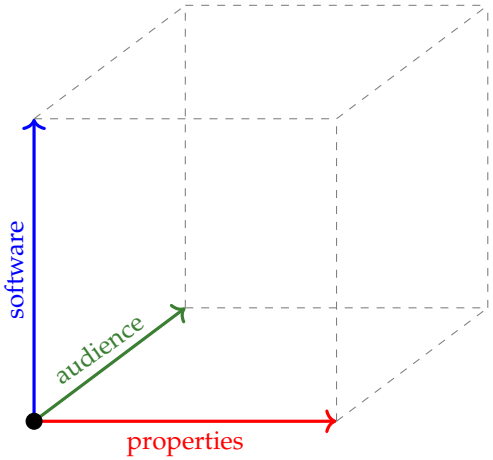


Figure 1.2: Incomplete approximation.

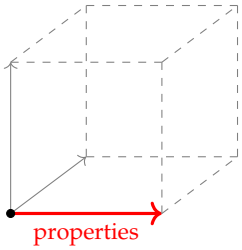
The overwhelming majority of the research in static analysis in the literature has focused on *safety correctness properties*, e.g., absence of runtime errors (such as divisions by zero, out-of-bound memory accesses, or assertion violations). This focus is well-justified for safety-critical software, where correctness violations can lead to catastrophic failures. However, modern software systems increasingly demand guarantees

that go beyond safety, and the demand for these guarantees has been democratized beyond traditional safety-critical domains. There is a growing need for static analyses that can prove more advanced *properties* of programs, as well as for static analyses that are more general-purpose, that is, applicable to a variety of *software* programs, and accessible to a broad *audience* rather than limited to experts.



Over the course of my research career, I have made contributions in all of these three directions. Most of my publications comprise solid theoretical contributions and rigorous practical implementation efforts.

Liveness Properties. With my doctoral dissertation [phd], I broadened the scope of static analysis by abstract interpretation to *termination* [c2] and other *liveness properties* [c5] of programs. The main insight that made this work possible was to relax some of the requirements imposed to the extrapolation technique (called *widening*) traditionally used to enforce convergence of the analysis, and allow the analysis to temporarily explore incorrect approximations of the behavior of a program [c11]. I implemented this work in the tool FuncTion [c6]. Additionally, I was able to transfer the main ideas at the root of this work into the context of software model checking, where I designed an approach for proving both termination and non-termination and implemented it in SeaHorn [c9].



[phd] Urban - Static Analysis by Abstract Interpretation of Functional Temporal Properties of Programs (École Normale Supérieure, 2015)
[c2] Urban - The Abstract Domain of Segmented Ranking Functions (SAS 2013)
[c5] Urban and Miné - Proving Guarantee and Recurrence Temporal Properties by Abstract Interpretation (VMCAI 2015)
[c6] Urban - FuncTion: An Abstract Domain Functor for Termination (TACAS 2015)
[c9] Urban at al. - Synthesizing Ranking Functions from Bits and Pieces (TACAS 2016).
[c11] Courant and Urban - Precise Widening Operators for Proving Termination by Abstract Interpretation (TACAS 2017)

Functional Properties. Building upon my doctoral work, I proposed a general static analysis framework for verifying *functional properties* of programs expressed in *computation tree logic* (CTL) [c15]. This generalization required rethinking the fixpoint computation strategy to account for the path quantifiers and temporal operators intrinsic to CTL (cf. Section 4.1).

Up to and including this work, I had only considered demonic non-determinism, i.e., assuming that non-deterministic program variables are controlled by an external adversary (e.g., attacker, scheduler, etc.). More recently, I became interested in studying different flavors of non-determinism, thus also considering uncontrolled non-deterministic variables (e.g., random seeds, etc.). With one of my Ph.D. student (Naïm Moussaoui Remil), I proposed a static analysis to automatically infer the minimal set of program variables that need to be controlled to ensure a program property expressed in CTL [c26]. This work opened up interesting connections to security, i.e., identifying attack vectors that

[c15] Urban et al. - Abstract Interpretation of CTL Properties (SAS 2018)
[c26] Moussaoui Remil et al. - Automatic Detection of Vulnerable Variables for CTL Properties of Programs (LPAR 2024)

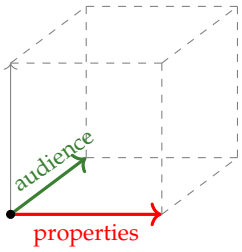
an adversary could exploit, and explainability, i.e., identifying which variables are critical to the manifestation of a particular program behavior, that I am currently exploring (cf. Section 4.2).

Hyperproperties. The shift to considering different kinds of non-deterministic program variables has driven this line of my research beyond safety and liveness functional properties, bringing me to currently investigate a broader range of program properties. These are *properties of sets of executions*, often called *hyperproperties*, including *termination resilience* – the impossibility for an adversary to cause definite non-termination of a program [u4] – and *functional non-exploitability* – the impossibility for an adversary to cause a program to violate or satisfy a functional property.

[c24] Campion et al. - Monotonicity and the Precision of Program Analysis (POPL 2024)
[u1] Campion et al. - Kernel Properties in Abstract Interpretation
[u4] Moussaoui Remil and Urban - Termination Resilience Static Analysis

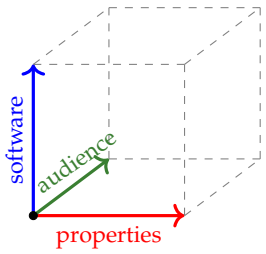
With one of my postdocs (Marco Campion) and other collaborators, I studied the impact of extensional program properties (i.e., relative to its observable input-output behavior) on the precision of a static analysis of the program, another important hyperproperty (called *completeness*), fundamental for understanding the limits of a static program analysis. In particular, we showed that a static analysis of programs (or program fragments) that manifest a monotone behavior will not produce false alarms [c24]. We further generalized this work showing that sufficient conditions for completeness arise from extensional program properties over a restricted subset of representative program inputs [u1].

Data Science Software. Several other compelling hyperproperties emerge in the context of data science software, notably software programs used to gather, triage, and prepare data within machine learning development pipelines. These programs – most often Python or R scripts – are written by domain experts rather than software engineers, making them especially prone to subtle errors. Supported by a Career Seed Grant awarded to me by ETH Zurich, I initiated the study of hyperproperties related to *data usage*, i.e., *how* input data is used by a program. In particular, I proposed an abstract interpretation framework for reasoning about dependencies between program variables, with the goal of detecting input data that remains unused by a program [c12]. This framework provides a unifying approach to harness various existing dependency-based analyses for data usage, such as non-interference analyses in security, strongly live variable analysis in compilers, and dependency analyses used in backward program slicing (cf. Section 4.3).



[c12] Urban and Müller - An Abstract Interpretation Framework for Input Data Usage (ESOP 2018)
[c27] Drobnjakovic et al. - An Abstract Interpretation-Based Data Leakage Static Analysis (TASE 2024)

More recently, together with collaborators at Microsoft Research, I generalized this framework to reason about dependencies between multi-dimensional program variables. I thus proposed an abstract interpretation-based static analysis to prove the *absence of data leakage*, which is an instance of data (non-)usage ensuring that datasets used for training and testing machine learning models remain independent [c27]. Data leakage can severely compromise the generalization capabilities of machine learning models, resulting in overly optimistic performance estimates during model evaluation, and leading to flawed decision-making with dangerous consequences when models are deployed in high-stakes applications.



Machine Learning Software. I studied another notable instance of data (non-)usage in the context of machine learning, notably applied to neural networks, to certify them to be free from algorithmic bias. Specifically, I

designed and developed a static analysis to prove that neural network predictions are independent from the values of certain sensitive input features [c17]. This property, called *dependency fairness*, is challenging to verify because it is a form of global robustness across the input space. What enabled a practical static analysis solution was a clever combination of forward and backward analyses to soundly parallelize and at the same time reduce the overall analysis effort (cf. Section 4.4). This work found a real-world application through a collaboration with Airbus, where we applied the analysis to certify dependency fairness of a neural network surrogate for aircraft braking distance estimation.
Through another industrial collaboration with Fujitsu, I was confronted with the inadequacy of ensuring robustness of the model prediction – a predominant focus in the literature in formal methods for machine learning – to provide guarantees of trustworthiness in practical applications. In response, we shifted the focus of the verification problem from the robustness of the model prediction to the *robustness of the explanation* of the model prediction [c21]. We proposed a first verification approach, while further more scalable approaches are ongoing work.

This work was pivotal in steering my research interests towards the application of formal methods to aid *machine learning explainability* [c23]. It has since become a major line of research in the SAIF project, which I lead within the national Priority Research Programme and Equipments on Artificial Intelligence (PEPR IA), funded by the France 2030 investment plan. Certified implementations of formal explainability techniques using the Rocq proof assistant are the object of a collaboration with the IRIT Computer Science Research Institute of Toulouse as part of the ForML project, funded by the French National Research Agency.

Motivated by the persistent divide between the formal methods and machine learning research communities, I have also been actively exploring how static analysis techniques can be leveraged *during the model training process* to achieve practical objectives of interest to the machine learning community, such as reducing the size of models [c19], or improving empirical robustness against adversarial attacks [j3].

[j3] De Palma et al. - On Using Certified-Training towards Empirical Robustness (TMLR 2025)
[c17] Urban et al. - Perfectly Parallel Fairness Certification of Neural Networks (OOPSLA 2020)
[c19] Ranzato et al. - Fairness-Aware Training of Decision Trees by Abstract Interpretation (CIKM 2021)
[c21] Munakata et al. - Verifying Attention Robustness of Deep Neural Networks against Semantic Perturbations (NFM 2023)
[c23] Pal et al. - Abstract Interpretation-Based Feature Importance for Support Vector Machines (VMCAI 2024)

Quantitative Properties. In several practical contexts, qualitative program properties – binary in nature, either satisfied or violated – are too strong. It is often acceptable or even expected to ask *how far* a property is from being satisfied or violated. For instance, it is well-known that completeness in static program analysis is extremely hard or even impossible to achieve in non-trivial cases. With one of my postdocs, we proposed a general framework for measuring the imprecision of static program analyses using pre-metrics [c22]. This quantitative approach enables a graded interpretation of the precision of a static analysis, called *partial completeness*: the static analysis is incomplete in general but its imprecision is sufficiently small to be acceptable for practical purposes. The use of pre-metrics in this work has inspired an ongoing line of research with other collaborators, studying novel interesting hyperproperties that arise from combining qualitative and quantitative approximations, such as *abstract Lipschitz continuity* [u2] and *partial abstract non-interference* [u3].

In parallel, with a recently graduated Ph.D. student (Denis Mazzucato), I proposed a static analysis framework to determine *to what degree* a program input contributes to its behavior [c25] – a quantitative form of

[c22] Campion et al. - A Formal Framework to Measure the Incompleteness of Abstract Interpretations (SAS 2023)
[c25] Mazzucato et al. - Quantitative Input Usage Static Analysis (NFM 2024)
[u2] Campion et al. - Abstract Lipschitz Continuity
[u3] Campion et al. - Measuring vs Abstracting: On the Relation between Distances and Abstract Domains

data usage. One instance of this framework, focusing on the impact of program inputs on the global number of loop iterations of the program, enabled us to formally prove that the real-world cryptographic library S2N-BIGNUM is immune to timing side-channel attacks [c28].

[c28] Mazzucato et al. - Quantitative Static Timing Analysis (SAS 2024)

1.2 Combining Static Analysis and Other Formal Methods

While the core of my research focuses on static analysis, I have also explored other formal methods and, in particular, their relation and the possible means of combining them with static analysis.

Software Model Checking. In parallel to my doctoral studies, with a collaborator, we used classic results in logic to establish precise correspondences between the mathematical foundations of abstract interpretation-based static analysis and model checking, with the objective of promoting the exchange of techniques between them and combining their complementary strengths [c8]. First, building on a classical result of Büchi that relates automata and logic, we encoded reachability as the satisfiability of formulas in a weak monadic second-order logic. Thus, we formally characterized static analyzers as solvers for the satisfiability of this family of formulas. Second, we gave a logical characterization of certain lattices used in static analyzers by using a construction of Lindenbaum and Tarski for generating lattices from logics. We thus showed that these lattices are subclassical fragments of first-order theories. This research, giving a logical characterization of abstract interpretation, led to practical applications in termination analysis: by integrating conflict-driven learning procedures – integral to the performance of SAT and SMT solvers used for model checking – we were able to drastically improve the precision of the FUNCTION termination static analyzer [c7].

[c7] D'Silva and Urban - Conflict-Driven Abstract Interpretation for Conditional Termination (CAV 2015)
[c8] D'Silva and Urban - Abstract Interpretation as Automated Deduction (CADE 2015)

Deductive Program Verification. Finally, during my postdoc at ETH Zurich, I worked on using static analysis to automatically infer specifications of (certain aspects of) the behavior of a program directly from its source code in order to reduce the effort required to formally verify that a program complies with its intended behavior. In particular, with other colleagues, we proposed a static analysis to automatically infer the memory footprint of an array-manipulating program, and produce specifications in the form of read and write access permissions that can be directly leveraged by deductive program verifiers [c13].

[c13] Dohrau et al. - Permission Inference for Array Programs (CAV 2018)

2

Collective Responsibilities

Responsabilités collectives

Throughout my academic career, I have devoted considerable effort to the collective functioning of the scientific community. These responsibilities, which complement my research and teaching activities, contribute to the organization and animation of our discipline. They also reflect my commitment to supporting others and helping the community grow.

2.1 Selection and Evaluation Committees

I have been a member of several *hiring and selection committees* for academic positions in computer science. These include admissibility juries for research scientist positions (CRCN/ISFP) at Inria, and selection committees for assistant and associate professor positions at diverse institutions, both in France and abroad. I also served as a member of the Commission des Emplois Scientifiques at Inria Paris in 2021 and 2022.

In addition, I have contributed to the evaluation of doctoral candidates through service on several *Ph.D. defense committees*, in France and internationally. In some cases, I acted as an official *reviewer of Ph.D. manuscripts*.

These responsibilities testify to the trust placed in my judgment for evaluating candidates at critical stages of their academic careers.

Hiring and Selection Committees

- Inria de l'Université de Lorraine (2025)
- Université de Lille (2025)
- Université de La Réunion (2024)
- École Polytechnique (2024)
- University of Copenhagen (2023)
- Inria Paris (2022)

Ph.D. Jury Member

- Linpeng Zhang (UK, 2025)
- John Törnblom (Sweden, 2025)
- Olivier Martinot (France, 2024)
- Guillaume Vidot (France, 2022)
- Guillaume Girol (France, 2022)
- Julien Girard-Satabin (France, 2021)
- Emilio Incerto (Italy, April 2019)

Ph.D. Manuscript Reviewer

- Pankaj Kumar Kalita (India, 2025)
- Marco Zanella (Italy, 2021)

2.2 Organization of Scientific Events

I have been involved in the organization of a number of scientific conferences, workshops, and invitational seminars. I am serving as *general chair* of the 20th International Conference on Integrated Formal Methods (iFM 2025), and I served as co-*chair of the program committee* of the 29th Static Analysis Symposium (SAS 2022) and the 10th Workshop on the State of the Art in Program Analysis (SOAP 2021). I have thus been a *member of the steering committee* of SOAP (from 2022 to 2024) and SAS (from 2023 until 2028). I have contributed as organizer of the posters and student research competition tracks at SPLASH 2022 and SPLASH 2021-2022, respectively. I have also organized scientific seminars (Dagstuhl Seminar 25421) and thematic workshops (N40AI at POPL 2024). I acted as coordinator for Dagstuhl Seminar 16471.

I also serve as a *member of the executive board* of ETAPS since 2019, where I am responsible for the Ph.D. activities of ETAPS. In particular, I started and I chair the annual ETAPS doctoral dissertation award since 2020. In addition, since 2023, I am the ETAPS representative on the

steering committee of the series of Summer Schools on Foundations of Programming and Software Systems (FoPSS).

2.3 Peer Review and Editorial Service

My involvement in *program committees* spans over 40 instances, across conferences – including flagship conferences such as POPL and CAV – as well as workshops, artifact evaluations, and competitions.

I frequently serve as a *reviewer* for international journals and conferences in programming languages, formal methods, and software engineering. I have reviewed book manuscripts (for MIT Press) and book chapters (for Springer) and acted as a remote referee for ERC proposals.

In addition to reviewing, I serve as *associate editor* for the journal Transactions on Programming Languages and Systems (TOPLAS) since 2023, and I served as *guest editor* for two special issues of the journal Formal Methods in System Design (FMSD) [Albarghouthi24, Pichardie25].

2.4 Mentoring Initiatives

I consider mentoring an essential and rewarding part of academic life. It is not only a way to support individual growth of early-career researchers, but also a key responsibility in fostering a thriving, inclusive, and supportive research community.

I co-organized several *mentoring workshops* at ETAPS, CAV, and FLoC. I have also been invited as a *panelist* at career development and networking events at SPLASH and ETH Zurich. I participate as a *mentor* in the SIGPLAN-M long-term mentoring program.

Each student at the École Normale Supérieure is supported by a *tutor* who provides guidance throughout their studies and helps them shape their academic projects. Since joining Inria in 2019 and becoming a permanent member of ENS, I have actively taken part in tutoring students.

2.5 Other Service

I participated in the *ethical review committee* of the 35th Conference on Neural Information Processing Systems (NeurIPS 2021).

I have taken up *publicity chair* roles for the 24th Static Analysis Symposium (SAS 2017) and 25th Static Analysis Symposium (SAS 2018), as well as for the 9th Federated Logic Conference (FLoC 2026).

Since 2023, I am a *member of the scientific advisory board* of the Formal Methods Laboratory (LMF) of the Université Paris-Saclay.

Program Committee Member

- **Conferences:** OOPSLA 2026, SAS 2025, FoSSaCS 2025, POPL 2025, LPAR 2024, CAV 2024, TACAS 2024, CAV 2023, NFM 2023, ESOP 2023, ICTAC 2022, CAV 2022, POPL 2022, SBLP 2021, CAV 2021, NFM 2021, FAccT 2021, SAS 2020, VSTTE 2020, CAV 2020, ESOP 2020, VMCAI 2020, iFM 2019, EMSOFT 2019, LOPSTR 2019, CAV 2019, VMCAI 2019, EMSOFT 2018, iFM 2018, SAS 2018, CAV 2018, SAS 2017, SAS 2016, VMCAI 2016
- **Workshops:** SOAP 2020, TAPAS 2019, AVoCS 2019, NSV 2019, WST 2018, AVoCS 2018, HCVS 2018, SPLASH 2015 Demos
- **Artifact Evaluations:** PLDI 2019 Artifact Evaluation, POPL 2019 Artifact Evaluation, CAV 2015 Artifact Evaluation
- **Competitions:** ACM SRC 2022, SPLASH 2020 SRC, PLDI 2018 SRC, SV-COMP 2015

Reviewer

- **Journals**: FnTs (2025), CACM (2022), FMSD (2022), TOPLAS (2022), SPE (2021), TSE (2018), TSE (2018), TOPLAS (2017), FMSD (2017), TOPLAS (2016), Acta Informatica (2016), TOPLAS (2015)
- **Conferences:** OOPSLA 2022, FMCAD 2022, SAS 2021, POPL 2020, NFM 2019, POPL 2018, ESOP 2017, VMCAI 2017, LOPSTR 2016, FM 2016, NFM 2016, TACAS 2016, ASE 2015, SAS 2015, CAV 2015, CAV 2014, TCS 2014

[Albarghouthi24] Albarghouthi et al. - Preface of the special issue on the conference on Computer-Aided Verification 2020 and 2021 (FMSD, 2024)
[Pichardie25] Pichardie et al. - Preface of the special issue on the static analysis symposium 2020 and 2022 (FMSD, 2025)

Mentoring Workshop Co-Organizer

- ETAPS Mentoring Workshop 2024
- ETAPS Mentoring Workshop 2023
- FLoC Mentoring Workshop 2022
- ETAPS Mentoring Workshop 2022
- CAV Mentoring Workshop 2021

Panel Member

- PLMW @ SPLASH 2024
- W @ SPLASH 2022
- VMI Career Event @ ETH Zurich

3

Invited Talks

Conférences invitées

Unless logistically infeasible, I *never decline* an invitation to give a talk.

3.1 Conferences

- **Nov 2024**: 17th Conference on Informatics (Informatics 2024), Poprad, Slovakia.
  Title: "Formal Methods for Machine Learning"
- **Oct 2024**: 31st Static Analysis Symposium (SAS 2024), Pasadena, USA.
  Title: "Abstract Interpretation-Based Certification of Hyperproperties for High-Stakes Machine Learning Software"
- **Apr 2023**: 29th Symposium on Model Checking of Software (SPIN 2023), Paris, France.
  Title: "Interpretability-Aware Verification of Machine Learning Software"
- **Oct 2019**: 26th Static Analysis Symposium (SAS 2019), Porto, Portugal
  Title: "Static Analysis of Data Science Software"
  `https://youtu.be/DX_wOrq9J18`
- **Jan 2016**: Congrès SIF 2016, Strasbourg, France.
  Title: "Analyse Statique par Interprétation Abstraite de Propriétés Temporelles des Programmes"

3.2 Workshops and Working Groups

- **Oct 2024**: 10th Workshop on Numerical and Symbolic Abstract Domains (NSAD 2024), Pasadena, USA
  Title: "Abstract Domains for Machine Learning Verification"
- **Apr 2024**: 29th Journées Formalisation des Activités Concurrentes (FAC 2024), Toulouse, France
  Title: "Machine Learning Interpretability and Verification"
- **Jul 2022**: "Vistas in Verified Software" Workshop, "Verified Software" Programme, Isaac Newton Institute for Mathematical Sciences, UK (remote)
  Title: "Static Analysis for Data Scientists"
- **Jun 2022**: 11th Workshop on the State Of the Art in Program Analysis (SOAP 2022), San Diego, USA
  Title: "Static Analysis for Data Scientists"
- **May 2022**: 1st Symposium on Challenges of Software Verification (CSV 2022), Venice, Italy
  Title: "Static Analysis for Data Scientists"
- **Nov 2021**: Journées du GT Vérif 2021, ENS Paris-Saclay, Gif-sur-Yvette, France
  Title: "An Abstract Interpretation Recipe for Machine Learning Fairness"
- **Jul 2021**: 4th Workshop on Formal Methods for ML-Enabled Autonomous Systems (FoMLAS 2021), Los Angeles, USA (remote)
  Title: "An Abstract Interpretation Recipe for Machine Learning Fairness"
- **Jan 2021**: Lorentz Center Workshop "Robust Artificial Intelligence", Lorentz Center, The Netherlands (remote)
  Title: "Perfectly Parallel Fairness Certification of Neural Networks"
- **Jan 2021**: Lorentz Center Workshop "Robust Artificial Intelligence", Lorentz Center, The Netherlands (remote)

Title: "Formal Methods for Robust Artificial Intelligence: State of the Art"
https://www.youtube.com/watch?v=ayXLWs4G4RU

- **Jul 2020**: 2nd Workshop on Democratizing Software Verification (DSV 2020), Los Angeles, USA (remote)
  Title: "A Static Analyzer for Data Science Software"
  https://www.youtube.com/watch?v=f8Cjpt-rzxE&t=4374s
- **Nov 2013**: 2nd Workshop on Analysis and Verification of Dependable Cyber Physical Software (AVDCPS 2013), Changsha, China
  Title: "The Abstract Domain of Piecewise-Defined Ranking Functions"

3.3 Invitational Seminars

- **Jun 2024:** Dagstuhl Seminar 25242 "Testing Program Analyzers and Verifiers", Schloss Dagstuhl, Germany
- **Feb 2024:** Dagstuhl Seminar 25061 "Logic and Neural Networks", Schloss Dagstuhl, Germany
  Title: "Static Analysis Methods for Neural Networks"
- **Jul 2022:** Dagstuhl Seminar 22291 "Machine Learning and Logical Reasoning: The New Frontier", Schloss Dagstuhl, Germany
  Title: "Data Usage across the Machine Learning Pipeline"
- **Oct 2017:** Shonan Meeting 100 "Analysis and Verification of Pointer Programs", Shonan Village Center, Japan
  Title: "An Abstract Interpretation Framework for Input Data Usage"
- **Sep 2017:** Shonan Meeting 108 "Memory Abstraction, Emerging Techniques and Applications", Shonan Village Center, Japan
  Title: "An Abstract Interpretation Framework for Input Data Usage"
- **May 2016:** Dagstuhl Seminar 16201 "Synergies among Testing, Verification, and Repair for Concurrent Programs", Schloss Dagstuhl, Germany
  Title: "Bringing Abstract Interpretation to Termination and Beyond"
- **Aug 2014:** Dagstuhl Seminar 14352 "Next Generation Static Software Analysis Tools", Schloss Dagstuhl, Germany
  Title: "Automatic Inference of Ranking Functions by Abstract Interpretation"

3.4 Other Seminars

- **Jun 2025:** University of Parma, Parma, Italy
  Title: "Termination Resilience Static Analysis"
- **Feb 2024:** Airbus, Toulouse, France
  Title: "(Hyper)Safety Certification of Neural Network Surrogates for Aircraft Braking Distance Estimation"
- **Jun 2024:** Université de La Réunion, Saint-Denis, Réunion (remote)
  Title: "Abstract Interpretation"
- **Jun 2024:** Scientific Board Meeting, Inria Paris, Paris, France
  Title: "Formal Methods for Machine Learning ~~Verification~~"
- **Mar 2024:** Quarkslab, Paris, France
  Title: "Machine Learning Interpretability and Verification"
- **Mar 2023:** CEA-LIST, Palaiseau, France
  Title: "Interpretability-Aware Verification of Machine Learning Software"
- **Feb 2023:** Séminaire IRILL, Center for Research and Innovation on Free Software, Paris, France
  Title: "Interpretability-Aware Verification of Machine Learning Software"
- **May 2022:** La Demi-Heure de Science, Inria Paris, Paris, France
  Title: "Interprétation Abstraite des Réseaux de Neurones"

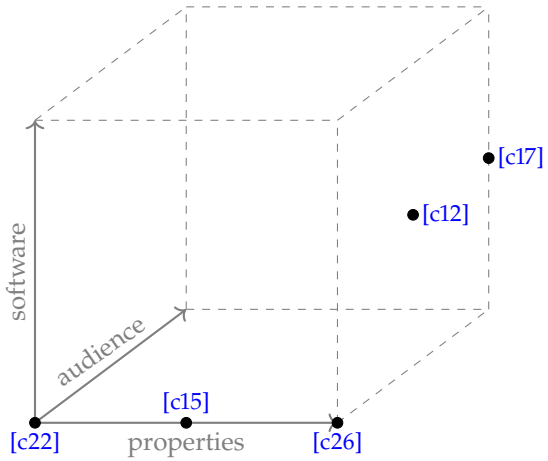
- **Nov 2021:** CEA-LIST, Palaiseau, France
  Title: "An Abstract Interpretation Recipe for Machine Learning Fairness"
- **May 2021:** École Normale Supérieure, Paris, France (remote)
  Title: "Perfectly Parallel Fairness Certification of Neural Networks"
- **Feb 2021:** Airbus, Toulouse, France (remote)
  Title: "Formal Methods for Robust Artificial Intelligence: State of the Art"
- **Nov 2020:** INSERM, Paris, France (remote)
  Title: "Static Analysis for Data Science"
- **Jun 2020:** Inria Rennes, Rennes, France (remote)
  Title: "Perfectly Parallel Fairness Certification of Neural Networks"
- **Jun 2020:** IRIF, Paris, France (remote)
  Title: "Perfectly Parallel Fairness Certification of Neural Networks"
- **May 2020:** Tel Aviv University, Tel Aviv, Israel (remote)
  Title: "Perfectly Parallel Fairness Certification of Neural Networks"
- **May 2020:** Thales Research & Technology, Palaiseau, France (remote)
  Title: "Perfectly Parallel Fairness Certification of Neural Networks"
- **Apr 2019:** Gran Sasso Science Institute (GSSI), L'Aquila, Italy
  Title: "What Programs Want: Automatic Inference of Input Data Specifications"
- **May 2018:** Inria Paris, Paris, France
  Title: "Static Program Analysis for a Software-Driven Society"
- **May 2018:** TU Wien, Vienna, Austria
  Title: "Static Program Analysis for a Software-Driven Society"
- **Mar 2018:** École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
  Title: "Static Program Analysis for a Software-Driven Society"
- **Mar 2018:** Stevens Institute of Technology, Hoboken, New Jersey, USA
  Title: "Static Program Analysis for a Software-Driven Society"
- **Mar 2018:** Max Planck Institute for Software Systems, Kaiserslautern, Germany
  Title: "Static Program Analysis for a Software-Driven Society"
- **Jan 2017:** Université Pierre et Marie Curie (Paris 6), Paris, France
  Title: "Synthesizing Ranking Functions from Bits and Pieces"
- **Aug 2015:** TU Wien, Vienna, Austria
  Title: "Abstract Interpretation as Automated Deduction"
- **Jul 2015:** SRI International, Menlo Park, USA
  Title: "Counterexample-Guided Inference of Ranking Functions"
- **Dec 2014:** University of Udine, Udine, Italy
  Title: "Proving Guarantee and Recurrence Temporal Properties by Abstract Interpretation"
- **Nov 2014:** ETH Zurich, Zurich, Switzerland
  Title: "Proving Guarantee and Recurrence Temporal Properties by Abstract Interpretation"
- **Oct 2014:** Queen Mary University of London, London, UK
  Title: "Automatic Inference of Ranking Functions by Abstract Interpretation"
- **Jun 2014:** University College London, London, UK
  Title: "Automatic Inference of Ranking Functions by Abstract Interpretation"
- **May 2014:** Inria Rennes, Rennes, France
  Title: "An Abstract Domain to Infer Ordinal-Valued Ranking Functions"
- **Mar 2014:** Inria Paris-Rocquencourt, France
  Title: "Automatic Inference of Ranking Functions by Abstract Interpretation"
- **Jan 2014:** IBM Thomas J. Watson Research Center, Yorktown Heights, USA
  Title: "Automatic Inference of Ranking Functions by Abstract Interpretation"
- **Nov 2013:** East China Normal University, Shanghai, China
  Title: "The Abstract Domain of Piecewise-Defined Ranking Functions"
- **Mar 2013:** University of Udine, Udine, Italy
  Title: "The Abstract Domain of Segmented Ranking Functions"

4

Critical Summary

Résumé critique

This section presents a critical summary of the following five scientific publications, selected for their significance within my research trajectory:



[c15] **Caterina Urban**, Samuel Ueltschi, Peter Müller. Abstract Interpretation of CTL Properties
In 25th Static Analysis Symposium (SAS 2018).
https://caterinaurban.github.io/publication/sas2018/

[c26] Naïm Moussaoui Remil, **Caterina Urban**, Antoine Miné. Automatic Detection of Vulnerable Variables for CTL Properties of Programs.
In 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2024).
https://inria.hal.science/hal-04710215

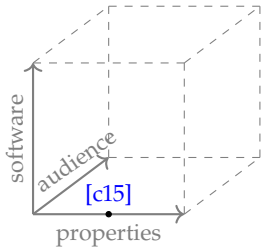
[c12] **Caterina Urban**, Peter Müller. An Abstract Interpretation Framework for Input Data Usage.
In 27th European Symposium on Programming (ESOP2018).
https://caterinaurban.github.io/publication/esop2018/

[c17] **Caterina Urban**, Maria Christakis, Valentin Wüstholz, Fuyuan Zhang. Perfectly Parallel Fairness Certification of Neural Networks.
In ACM on Programming Languages (PACMPL), Conference on Object-Oriented Programming Systems, Languages, and Applications 2020 (OOPSLA 2020).
https://inria.hal.science/hal-03091870

[c22] Marco Campion, **Caterina Urban**, Mila Dalla Preda, Roberto Giacobazzi. A Formal Framework to Measure the Incompleteness of Abstract Interpretations.
In 30th Static Analysis Symposium (SAS 2023).
https://inria.hal.science/hal-04249990

Each publication is contextualized and accompanied by a critical reflection on the scientific choices made, the results obtained and their impact, the limitations encountered, as well as the research questions raised and the directions opened for future work.

4.1 Abstract Interpretation of CTL Properties



[c15] Urban et al - Abstract Interpretation of CTL Properties (SAS 2018)

Computation tree logic (CTL) is a temporal logic introduced by Clarke and Emerson to overcome certain limitations of linear temporal logic (LTL) for program specification purposes. *With [c15], I substantially generalized my doctoral thesis*, introducing a static analysis framework based on abstract interpretation for verifying functional properties of programs expressed in CTL. This work addresses limitations of previous approaches in the literature which were often restricted to finite-state programs (or only certain classes of infinite-state program), or limited in scope to subsets of CTL without existential quantifiers (or only supporting an indirect treatment of existential quantifiers).

Central to the approach is the abstraction of the operational semantics of a given program into a function mapping program states to ordinals, bounding the number of program execution steps needed to satisfy a given CTL formula. This semantics is defined inductively on the structure of the CTL formula, enabling a principled and *compositional* treatment of functional program properties with arbitrary alternations of universal and existential quantifiers, and unifying reasoning about safety and liveness properties in a single formalism, i.e., leveraging ranking functions for liveness properties. Further decidable approximations are derived by leveraging the abstract domain based on *piecewise-defined functions* [c4] developed as part of my doctoral thesis, augmented with *under-approximating operators* [Miné12] to handle CTL formulas with existential quantifiers. A piecewise-defined function for a CTL formula is automatically inferred through backward static analysis of the given program by building upon the piecewise-defined functions for its subformulas. It over-approximates the value of the corresponding concrete semantics and, by under-approximating its domain of definition, yields a *sufficient precondition* for the CTL formula. The analysis thus provides actionable insights even when properties are only conditionally satisfied.

[c4] Urban and Miné - A Decision Tree Abstract Domain for Proving Conditional Termination (SAS 2014)
[Miné12] Miné - Inferring Sufficient Conditions with Backward Polyhedral Under-Approximations (2012)

Let us consider the program snippet in Figure 4.1 and the CTL formula $\mathsf{AG}(x = 1 \Rightarrow \mathsf{AF}(x = 0))$ stating that it is always the case (AG) that whenever the lock is acquired ($x = 1$) it will always eventually (AF) be released ($x = 0$). The static analysis proposed in [c15] automatically infers the following piecewise-defined ranking function at program point 4:

$$\lambda x n.\ \begin{cases} 0 & x = 0 \\ 2 & x \neq 0 \wedge n \leq 0 \\ 2n + 2 & \text{otherwise} \end{cases}$$

Its value indicates the maximum number of program execution steps needed to reach the next state where the lock is released, i.e., the next state that satisfies $x = 0$. The function inferred at the beginning of the program, program point 1, is only defined when $x \neq 1$ (i.e., the lock is not

```
while [1]( rand() ) {
  [2]x := 1
  [3]n := rand()
  while [4]( n > 0 ) {
    [5]n := n − 1
  }
  [6]x := 0
}
while [7]( true ) {}[8]
```

Figure 4.1: Standard lock acquire/release-style program [Cook12], where rand() is a random number generation function. Assignments x := 1 and x := 0 are acting as acquire and release, respectively

[Cook12] Cook et al - Temporal Property Verification as a Program Analysis Task (2012)

already acquired initially), yielding the weakest sufficient precondition for the CTL formula $\mathsf{AG}(x = 1 \Rightarrow \mathsf{AF}(x = 0))$.

The approach is implemented in FuncTion and the experimental evaluation in [c15] showed its effectiveness on a wide variety of benchmarks. Nonetheless, the precision of the static analysis is sensitive to the employed widening heuristic [c11], and can degrade considerably due to unfortunate interactions between under-approximations needed for existential CTL formulas and non-deterministic variable assignments.

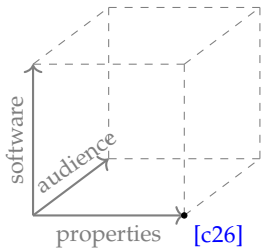
The framework is expressive enough to support further extensions toward LTL or CTL*. However, accommodating these logics would require the integration of some form of trace partitioning [Rival07], since the interpretation of LTL formulas is defined in terms of program executions rather than program states as CTL.

This work served as a starting point and building block for the Ph.D. thesis of Naïm Moussaoui Remil. This is further discussed in the next section.

[c11] Courant and Urban - Precise Widening Operators for Proving Termination by Abstract Interpretation (TACAS 2017)
[Rival07] Rival and Mauborgne - The Trace Partitioning Abstract Domain (2007)

4.2 Automatic Detection of Vulnerable Variables for CTL Properties of Programs

This work departs from the usual setting in which all program variables are treated equally, and instead adopts a refined perspective that distinguishes between variables that are *under the control of an external adversary* and those that remain *uncontrolled*. Such distinction was considered by earlier work [Girol21, Parolini24] but limited to the analysis of safety program properties, while [c26] extends this refined perspective to a much broader class of functional program properties. Specifically, [c26] proposes a static analysis that automatically identifies sets of vulnerable program variables, that is, subset-*minimal sets of program variables that need to be controlled* to ensure the satisfaction of a functional program property expressed in CTL. The analysis is a principled way to reason about attacker capabilities, agnostic to the underlying functional verification method, and [c26] presents an instance built on top of the static analysis framework for CTL discussed in the previous section [c15]. Thus, as a by-product, the approach also infers sufficient preconditions over the vulnerable variables that guarantee the satisfaction of a given CTL formula.



[c26] Moussaoui Remil et al. - Automatic Detection of Vulnerable Variables for CTL Properties of Programs (LPAR 2024)

[Girol21] Girol et al. - Not All Bugs Are Created Equal, But Robust Reachability Can Tell the Difference
[Parolini24] Parolini and Miné - Sound Abstract Nonexploitability Analysis (VMCAI 2024)

Let us consider the program snippet on the right and the CTL formula $\mathsf{AF}(3 : \text{true})$ stating that the error location at program point 3 is always eventually (AF) reached. The static analysis proposed in [c26] automatically infers that $\{y, z\}$ and $\{x\}$ are alternative sets of vulnerable variables: indeed, it is enough to control the value of y and z (such that $y \leq z$ and the loop is never entered), or to control the value of x (such that $x \geq 0$ and the loop always terminates) to ensure the reachability of the error location *independently of the values of the uncontrolled variables*.

```
while 1( y > z ) {
  2y := y - z
}
3 // error
```

This work constitutes the first contribution made by my Ph.D. student Naïm Moussaoui Remil. He is currently addressing the limitation of the approach to numerical programs without pointers or memory manipulations. Specifically, he is developing an extension of the underlying static analysis framework for CTL [c15] to handle pointer-manipulating programs. This enhancement is essential for broadening the applicability of the approach proposed in [c26] to more realistic and complex programs,

notably in security, where vulnerable variable sets would identify attack vectors that could compromise a system.

More fundamentally, the distinction between controlled and uncontrolled variables has deep semantic implications, as it enables the coexistence of different flavors of non-determinism within the same program: demonic non-determinism, associated with adversarial control, and angelic non-determinism, reflecting benign or arbitrary behavior. This gives rise to interesting program hyperproperties such as *termination resilience* – the impossibility for an adversary to cause definite non-termination of a program independently of the value of the uncontrolled variables [u4] – and *functional non-exploitability* – the impossibility for an adversary to cause a program to violate or satisfy a functional property (generalizing safety non-exploitability [Parolini24]).

The identification of vulnerable variables can be interpreted as a form of causal attribution – highlighting which program variables are responsible for the satisfaction or violation of a given property under adversarial influence. This opens promising connections with *causality* and *responsibility analysis* [Deng19] that are worth exploring. It also aligns with current trends in logic-based explainability, where formal reasoning techniques are used to provide minimal sufficient explanations of the behavior of a system [Marques-Silva24]. In this light, vulnerable variable sets and their associated preconditions offer interpretable evidence for understanding why a program exhibits a certain behavior.

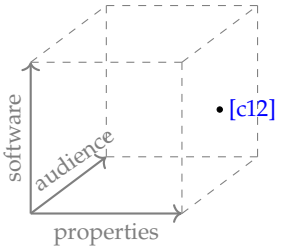
[u4] Moussaoui Remil and Urban - Termination Resilience Static Analysis
[Parolini24] Parolini and Miné - Sound Abstract Nonexploitability Analysis (VMCAI 2024)
[Deng19] Deng and Cousot - Responsibility Analysis by Abstract Interpretation (SAS 2019)
[Marques-Silva24] Marques-Silva - Logic-Based Explainability: Past, Present and Future (ISoLA 2024)

4.3 An Abstract Interpretation Framework for Input Data Usage

This work represents my very first step in *democratizing static analysis* towards a different kind of software programs – *data science software* – and targeting a different audience – *data scientists*. It was motivated by the pervasiveness of data-driven decision-making software in many domains, ranging from retail, to manufacturing, finance, and even healthcare.

In [c12], I focused on leveraging static program analysis to detect program flaws that do not cause failures. Such silent mistakes can have serious consequences since code that produces an erroneous but plausible result gives no indication that something went wrong. In particular, I proposed an abstract interpretation framework to automatically detect input *data that remains unused* by a program.

Central to the framework is a characterization of when a program uses (some of) its input data by means of a notion of *dependency* between the input data and the outcome of the program. Its definition, shown in Figure 4.2, formalizes the idea that a program input i is used if a certain outcome of the program (σ_ω) is not possible when the input variable i has a certain value (v). Unlike the usual definitions – notably in the secure information flow literature – this definition formalizes that changing the value of i affects the outcome of the program in a way that also accounts for non-termination (since the outcome σ_ω is either a final state or non-termination) as well as non-determinism (via the universal quantification over all program execution traces σ' where i has initial value v).



[c12] Urban and Müller - An Abstract Interpretation Framework for Input Data Usage (ESOP 2018)

$$\text{USED}_i \stackrel{\text{def}}{=} \exists \sigma v \colon A \wedge \forall \sigma' \colon B \Rightarrow C$$
$$A \stackrel{\text{def}}{=} \sigma_0(i) \neq v$$
$$B \stackrel{\text{def}}{=} \sigma_0 \equiv_{\setminus i} \sigma'_0 \wedge \sigma'_0(i) = v$$
$$C \stackrel{\text{def}}{=} \sigma_\omega \neq \sigma'_\omega$$

Figure 4.2: Definition of when a program uses an input variable i, where σ, σ' are execution traces of the program, v is a value for i, σ_0 and σ_ω represent the initial state of a trace and its outcome (a final state or non-termination), and $\sigma_0 \equiv_{\setminus i} \sigma'_0$ denotes initial states of program traces only differing on the value of i.

This key definition encompasses notions of dependencies that arise in many different contexts, such as secure information flow, program slicing, and provenance or lineage analysis. It thus provides a *unifying framework* for reasoning about existing analyses based on dependencies and their applicability to detect unused input data. In [c12], I surveyed non-interference [Assaf17] and strongly-live variable [Giegerich81] analyses and identified key design decisions that hinder or facilitate their applicability. I additionally proposed a more precise static analysis based on *syntactic dependencies* between program variables, accompanied by an implementation targeting Python programs in the open-source tool Lyra.

[Assaf17] Assaf et al. - Hypercollecting Semantics and Its Application to Static Analysis of Information Flow (POPL 2017)
[Giegerich81] Giegerich et al. - Invariance of Approximate Semantics with Respect to Program Transformations (1981)

This foundational work has catalyzed a large body of subsequent research, spanning both theoretical developments and practical applications.

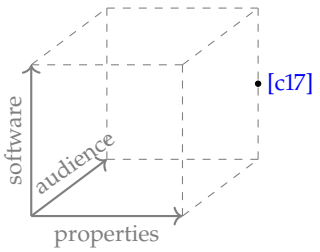
First of all, it led to the Ph.D. thesis of Denis Mazzucato, which introduced a *quantitative generalization* of input data usage aimed at assessing the extent to which a program input influences its behavior [c25]. One notable instance of this quantitative framework focused on the impact of program inputs on the global number of program loop iterations [c28]. Its practical implementation, building upon the syntactic dependency analysis proposed in [c12], enabled proving the immunity to timing side-channel attacks of the real-world cryptographic library S2N-Bignum.

[c25] Mazzucato et al. - Quantitative Input Usage Static Analysis (NFM 2024)
[c27] Drobnjakovic et al. - An Abstract Interpretation-Based Data Leakage Static Analysis (TASE 2024)
[c28] Mazzucato et al. - Quantitative Static Timing Analysis (SAS 2024)
[w5] Dolcetti et al. - Towards a High Level Linter for Data Science (NSAD 2024)

Other practical applications of [c12] emerged from instances and generalization of the input-outcome dependency definition in Figure 4.2. An instance applied to neural networks enabled proving *absence of algorithmic bias*, and is further discussed in the next section. A generalization to multi-dimensional program variables led to an abstract interpretation framework for proving *absence of data leakage* between data used for training and testing machine learning models [c27].

Moreover, ongoing work on an abstract interpretation-based linter tool for data science practitioners [w5], builds upon Lyra.

4.4 Perfectly Parallel Fairness Certification of Neural Networks



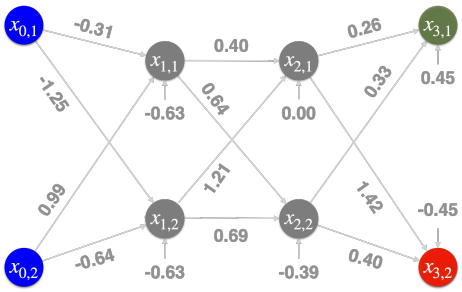
[c17] Urban et al. - Perfectly Parallel Fairness Certification of Neural Networks (OOPSLA 2020)

A number of cases over the years have shown that machine-learned systems may reproduce or exacerbate bias explicitly or implicitly embedded in their training data [Larson16, Obermeyer19]. In response to these concerns, the European Commission introduced the Artificial Intelligence Act in April 2021, which establishes a comprehensive legal framework to govern the development and deployment of machine learning systems, with particular emphasis on preventing discriminatory outcomes. Within this regulatory landscape, [c17] proposes a valuable static analysis for assessing fairness of neural network classifiers.

[Larson16] Larson et al. - How We Analyzed the COMPAS Recidivism Algorithm (2016)
[Obermeyer19] Obermeyer et al. - Dissecting Racial Bias in an Algorithm Used to Manage the Health of Populations (2019)

The approach is tailored for *dependency fairness* [Galhotra17], a form of global robustness requiring the neural network classification to be independent of the values of the chosen sensitive input features. Formally, the definition of (dependency) bias with respect to a sensitive feature i, shown in Figure 4.3, formalizes the idea that the neural network can yield different predictions (σ_ω and σ'_ω) for input data that only differs in the value of i. It is an instance of the definition in Figure 4.2, simplified since trained neural networks are deterministic and always terminating.

Let us consider the following toy neural network binary classifier:



It comprises two input neurons $x_{0,1}$ (credit amount) and $x_{0,2}$ (age of the person requesting the credit), and two output neurons $x_{3,1}$ (request approved) and $x_{3,2}$ (request denied). In between there are two hidden layers, each with two hidden neurons. Each hidden or output neuron computes an affine combination of the values of the neurons in the previous layer using the weights on the edges. Hidden neurons additionally apply an activation function such as the ReLU (i.e., ReLU(x) = max(0, x)). We say that the neuron is active when $x > 0$ before applying the ReLU, and inactive when $x \leq 0$. The prediction of the neural network for given values of the input neurons is the output neuron with the maximum value. Assuming that input data is normalized in the range 0 to 1, the static analysis approach proposed in [c17] automatically infers that the neural network discriminates with respect to the age of the person if the requested credit amount is larger than 0.53, which amounts to ~47% of the entire input space of the neural network.

$$\text{BIAS}_i \stackrel{\text{def}}{=} \exists \sigma \sigma' : B \Rightarrow C$$

$$B \stackrel{\text{def}}{=} \sigma_0 \equiv_{\setminus i} \sigma'_0$$

$$C \stackrel{\text{def}}{=} \sigma_\omega \neq \sigma'_\omega$$

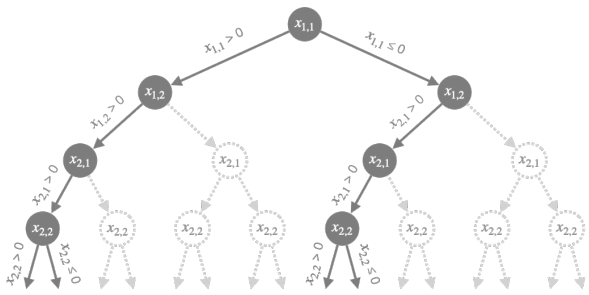
Figure 4.3: Definition of when a neural network is biased with respect to a sensitive input feature i, where σ_0 and σ_ω are the input data fed to the neural network and its prediction, and $\sigma_0 \equiv_{\setminus i} \sigma'_0$ denotes input data only differing on i.

[Galhotra17] Galhotra et al. - Fairness Testing: Testing Software for Discrimination (FSE 2017)

The approach is a clever *combination of a forward and a backward analyses* that enables perfect parallelization and a reduction in the overall analysis effort. At its core, the forward analysis employs an abstract domain to over-approximate the set of possible values that each neuron can take, and iteratively divides the input space of the neural network into independent partitions that satisfy the configured resource limits. These constrain the size of a partition with a lower bound L, and set an upper bound U on the number of hidden neurons with unknown activation status (neither always active or always inactive) found by the analysis of the partition. It turns out that, often, *multiple input partitions are associated to the same activation pattern*. In the toy example above, both partitions I_1 and $I_{2,1}$ (cf. the note on the right) have $x_{1,1}$ and $x_{2,2}$ with unknown activation status, as well as $x_{1,2}$ and $x_{2,1}$ always active.

A forward analysis of the toy neural network above with the interval abstract domain (tracking the range of possible values for each neuron) and constrained by a lower bound of 0.25 and an upper bound of 2 iteratively partitions the input space I ($x_{0,1} \in [0, 1]$ and $x_{0,2} \in [0, 1]$) of the neural network into I_1 ($x_{0,1} \in [0, 0.5]$ and $x_{0,2} \in [0, 1]$), $I_{2,1}$ ($x_{0,1} \in [0.5, 0.75]$ and $x_{0,2} \in [0, 1]$) and $I_{2,2}$ ($x_{0,1} \in [0.75, 1]$ and $x_{0,2} \in [0, 1]$). The analysis finds that all hidden neurons have unknown activations status for I – then split into I_1 and I_2 ($x_{0,1} \in [0.5, 1]$ and $x_{0,2} \in [0, 1]$) – and I_2 – then split into $I_{2,1}$, $I_{2,2}$.

The backward analysis uses the abstract domain of polyhedra (tracking conjunctions of linear constraints over the neurons) to identify the subregions of the input partitions that correspond to each class prediction. It leverages the activation patterns to prune unfeasible executions.



The possible classification outcomes of the toy neural network above are represented by the constraints $x_{3,1} > x_{3,2}$ (credit request approved) and $x_{3,2} > x_{3,1}$ (credit request denied). For the input partitions I_1 and $I_{2,1}$, the analysis of the hidden layers can leverage their activation pattern to prune away the dotted execution paths on the left (since $x_{1,2}$ and $x_{2,1}$ are always active).

Finally, each identified subregion havocs any constraint on the sensitive input features and the analysis check for intersections between subregions corresponding to different class predictions. Any non-empty intersection is a *witness* of bias of the neural network: the input data in the intersection is classified differently depending on the value of the sensitive input features. If no intersection can be found then all identified subregions are certified to be fair. The analysis of the toy example above, is able to certify I_1 to be fair and, instead, finds bias in $I_{2,1}$, for $x > 0.53$, and in $I_{2,2}$.

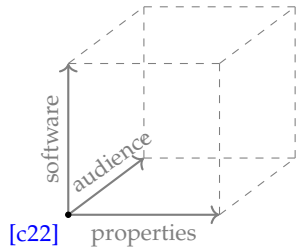
The approach is implemented in the open-source tool Libra. In follow-up work, I proposed improvements to the forward analysis. Together with my Ph.D. student Denis Mazzucato, we obtained over 10% of improvement in precision with a *reduced product* abstract domain [c18]. With my Ph.D. student Serge Durant, we improved scalability with a *smarter partitioning heuristic* that leverages the underlying abstract domain to decide how a partition is divided [w3]. This heuristic yielded a reduction of 1 to 2 orders of magnitude in the number of input partitions produced by the analysis, significantly decreasing its computational cost.

[c18] Mazzucato and Urban - Reduced Products of Abstract Domains for Fairness Certification of Neural Networks (SAS 2021)
[w3] Durand et al. - ReCIPH: Relational Coefficients for Input Partitioning Heuristic (WFVML 2022)

While scalability remains limited to neural networks with low dimensional inputs and hundreds of hidden neurons, the approach is nevertheless sufficiently effective to handle real-world applications. In a collaboration with Airbus, I successfully used the static analysis proposed in [c17] and improved in [c18, w3] to certify dependency fairness of a neural network surrogate for aircraft braking distance estimation (e.g., proving that estimating the braking distance in the expected altitude range does not alter the predicted runway overrun risk).

4.5 A Formal Framework to Measure the Incompleteness of Abstract Interpretations

This work tackles the challenge of reasoning rigorously about *incompleteness* in abstract interpretation, the inevitable loss of precision caused by the abstraction [Giacobazzi15]. It propose a formal framework for quantitatively measuring imprecision – thus enabling a principled way to reason about and compare incomplete static analyses.



[c22]

[c22] Campion et al. - A Formal Framework to Measure the Incompleteness of Abstract Interpretations (SAS 2023)

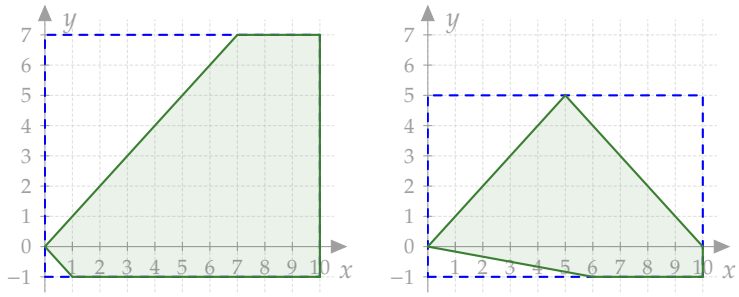
In particular, [c22] generalizes the notion of *partial completeness* introduced by [Campion22] using *pre-metrics* compatible with the partial order underlying the abstraction. Let $\langle C, \preccurlyeq \rangle$ be a partially ordered set. A pre-metric $\delta \colon C \times C \to \mathbb{R} \cup \{\infty\}$ is non-negative ($\forall x, y \in C \colon \delta(x, y) \geq 0$) and satisfies the if-identity axiom ($\forall x, y \in C \colon x = y \Rightarrow \delta(x, y) = 0$). It is compatible with $\langle C, \preccurlyeq \rangle$ if it is meaningful for comparing elements on the same chain in C, i.e., it additionally satisfies $\forall x, y, z \in C \colon x \preccurlyeq y \preccurlyeq z \Rightarrow \delta(x, y) \leq \delta(x, z) \wedge \delta(y, z) \leq \delta(x, z)$. Pre-metrics broaden the applicability of partial completeness to concretization-based abstractions, while the original definition in [Campion22] is limited by the requirement of having a Galois connection. The generalized definition of partial completeness is shown in Definition 4.5.1. The distance δ between $f \circ \gamma$ and $\gamma \circ f^\natural$ is a measure of the imprecision introduced by $f^\natural$ with respect to f, and the definition requires this imprecision to be bounded by ε.

[Giacobazzi15] Giacobazzi et al. - Analyzing Program Analyses (POPL 2015)
[Campion22] Campion et al. - Partial (In)Completeness in Abstract Interpretation: Limiting the Imprecision in Program Analysis (POPL 2022)

Definition 4.5.1 (Partial Completeness) *Given pre-ordered sets $\langle C, \preccurlyeq \rangle$ and $\langle A, \leq \rangle$ related by a concretization function $\gamma \colon C \to A$ and a pre-metric δ compatible with $\langle C, \preccurlyeq \rangle$, a function $f^\natural \colon A \to A$ is an* ε-partial complete approximation *of a function $f \colon C \to C$ if and only if $\forall x \in A \colon \delta(f(\gamma(x)), \gamma(f^\natural(x))) \leq \varepsilon$.*

Let us consider the program P in Figure 4.4. The best linear convex approximation of the loop invariant at program point 2 is $0 \leq y \leq x \wedge x + y \leq 10$

represented by the green triangle in Figure 4.5. We measure the imprecision introduced by a numerical static analysis with the difference in percentage between the volumes of the hyperrectangles enclosing the loop invariants (shown in dashed blue for the best invariant in Figure 4.5). These are the loop invariants inferred by INTERPROC using the octagons [Mine01] (left) and polyhedra [Cousot78] (right) abstract domains:



The static analysis using octagons is 60-partial complete: the hyperrectangle enclosing the inferred invariant has 60% more volume than the one in Figure 4.5. Instead, the analysis using polyhedra is 20-partial complete. The generality of the framework proposed [c22] also allows measuring the relative precision of static analyses: in this case, the analysis using octagons is 33.33-partial complete with respect to using polyhedra.

This work led to a fruitful ongoing collaboration centered on the study of novel interesting hyperproperties that arise from combining qualitative approximations – modeled using upper closure operators [Cousot79] – and quantitative approximations – expressed via pre-metrics. We are currently studying *abstract Lipschitz continuity* [u2] – a generalization of Lipschitz continuity that ensures that small differences in *semantic approximations* of inputs to a function (e.g., a program) lead to proportionally bounded differences in the *semantic approximations* of its outputs – and *partial abstract non-interference* [u3] – a generalization of abstract non-interference [Giacobazzi04] that admits a bounded error in output.

```
[1] x := 0
[2] y := 0
while [2]( x ≤ 9 ∧ y ≥ 0 ) {
  [3]if ( x ≤ 4 ) {
    [4]y := y + 1
  } else {
    [6]y := y - 1
  }
  [6]x := x + 1
}
```

Figure 4.4: The program P [c22].

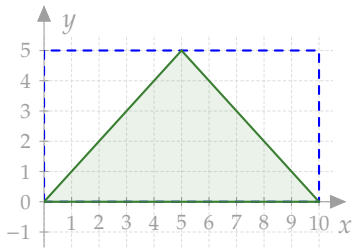


Figure 4.5: The best convex approximation of the loop invariant of program P.

[Mine01] Miné - A New Numerical Abstract Domain Based on Difference-Bound Matrices (PADO 2021)
[Cousot78] Cousot and Halbwachs - Automatic Discovery of Linear Restraints Among Variables of a Program (POPL 1978)
[Cousot79] Cousot and Cousot - Systematic Design of Program Analysis Frameworks (POPL 1979)
[Giacobazzi04] Giacobazzi and Mastroeni - Abstract Non-Inteference: Parametrizing Non-Interference by Abstract Interpretation (POPL 2004)
[u2] Campion et al. - Abstract Lipschitz Continuity
[u3] Campion et al. - Measuring vs Abstracting: On the Relation between Distances and Abstract Domains

RÉSUMÉ

Ce document présente un aperçu de mon parcours de recherche, qui vise à améliorer la qualité et la fiabilité des systèmes logiciels modernes grâce à des analyses statiques avancées. Ancrés dans la théorie de l'interprétation abstraite, mes travaux élargissent le champ de l'analyse statique des programmes pour couvrir une gamme variée de propriétés fonctionnelles, hyperpropriétés et propriétés quantitatives. Alliant contributions théoriques et développement d'outils pratiques, ma recherche propose des méthodes d'analyse statique pour vérifier des spécifications en logique temporelle, détecter les variables vulnérables dans des contextes adverses, ainsi que quantifier l'imprécision des analyses et comparer différentes techniques d'analyse. Elle ouvre également la voie à des analyses statiques adaptées à de nouveaux publics — tels que les data scientists — et à de nouveaux domaines logiciels — tels que les chaînes de développement en apprentissage automatique. L'ambition qui sous-tend ce travail est de rendre l'analyse statique plus expressive, accessible et en phase avec l'évolution des besoins des logiciels, de leurs développeurs et de leurs utilisateurs.

ABSTRACT

This documents presents an overview of my research journey aimed at enhancing the quality and reliability of modern software systems through advanced static analyses. Rooted in the theory of abstract interpretation, my work broadens the scope of static program analysis to address a diverse range of functional properties, hyperproperties, and quantitative properties. Through a blend of theoretical contributions and practical tool development, my research contributes static analysis methods for verifying temporal logic specifications, detecting vulnerable variables in adversarial settings, as well as quantifying analysis imprecision and comparing static analyses. It also pioneers static analyses tailored to new audiences -- such as data scientists -- and software domains -- such as machine learning development pipelines. The overarching vision is to make static analysis more expressive, accessible, and aligned with the evolving needs of software, its developers, and its users.