

2025 Activity Report

RESEARCH CENTRE: Inria Paris Centre

IN PARTNERSHIP WITH: CNRS, Ecole normale supérieure de Paris

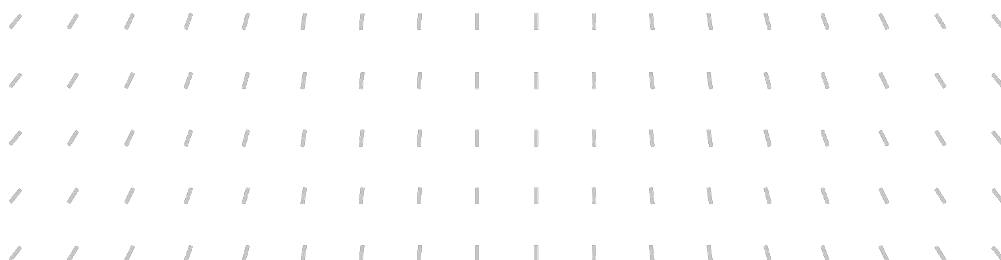

Project-Team

ANTIQUÉ

Static Analysis by Abstract Interpretation



In collaboration with Département d'Informatique de l'Ecole Normale Supérieure



Project-Team ANTIQUE

Creation of the Project-Team: 2015 April 01

Each year, Inria research teams publish an Activity Report presenting their work and results over the reporting period. These reports follow a common structure, with some optional sections depending on the specific team. They typically begin by outlining the overall objectives and research programme, including the main research themes, goals, and methodological approaches. They also describe the application domains targeted by the team, highlighting the scientific or societal contexts in which their work is situated. The reports then present the highlights of the year, covering major scientific achievements, software developments, or teaching contributions. When relevant, they include sections on software, platforms, and open data, detailing the tools developed and how they are shared. A substantial part is dedicated to new results, where scientific contributions are described in detail, often with subsections specifying participants and associated keywords. Finally, the Activity Report addresses funding, contracts, partnerships, and collaborations at various levels, from industrial agreements to international cooperations. It also covers dissemination and teaching activities, such as participation in scientific events, outreach, and supervision. The document concludes with a presentation of scientific production, including major publications and those produced during the year.

Keywords

Computer sciences and digital sciences

- A2. – Software sciences
 - A2.1. – Programming Languages
 - A2.1.1. – Semantics of programming languages
 - A2.1.6. – Concurrent programming
 - A2.1.7. – Distributed programming
 - A2.1.12. – Dynamic languages
 - A2.2.1. – Static analysis
- A2.3. – Embedded and cyber-physical systems
 - A2.3.1. – Embedded systems
 - A2.3.2. – Cyber-physical systems
 - A2.3.3. – Real-time systems
- A2.6.1. – Operating systems
- A4.4. – Security of equipment and software
- A4.5. – Formal method for verification, reliability, certification
 - A4.5.1. – Static analysis
 - A4.5.2. – Model-checking
 - A4.5.3. – Program proof

Other research topics and application domains

- B1.1. – Biology
 - B1.1.8. – Mathematical biology
 - B1.1.10. – Systems and synthetic biology
- B5.2. – Design and manufacturing
 - B5.2.1. – Road vehicles
 - B5.2.2. – Railway
 - B5.2.3. – Aviation
 - B5.2.4. – Aerospace
- B6.1. – Software industry
 - B6.1.1. – Software engineering
 - B6.1.2. – Software evolution, maintenance
- B6.6. – Embedded systems

Contents

Project-Team ANTIQUE	1
1 Team members, visitors, external collaborators	5
2 Overall objectives	6
2.1 Verifying safety critical systems	6
2.2 Formal definition of safety properties	6
2.3 Other verification approaches	7
2.4 Emergent applications	7
3 Research program	7
3.1 Semantics	7
3.1.1 The role of semantics in program verification	7
3.1.2 Understanding program semantics using category theory	8
3.2 Abstract interpretation and static analysis	8
3.3 Applications of the notion of abstraction in semantics	8
3.4 From properties to explanations	9
4 Application domains	10
4.1 Verification of safety critical embedded software	10
4.2 Static analysis of software components and libraries	11
4.3 Models of mechanistic interactions between proteins	11
4.4 Application of distributed algorithms	11
4.5 Formal reasoning about parallel program execution	12
4.6 Static analysis of data science and machine learning software	12
4.7 Probabilistic programming and statistical inference	13
5 Social and environmental responsibility	13
5.1 Impact of research results	13
6 Highlights of the year	13
6.1 Awards	13
6.2 Conference organisations	13
7 Latest software developments, platforms, open data	14
7.1 Latest software developments	14
7.1.1 Astrée	14
7.1.2 AstréeA	14
7.1.3 CESAn	15
7.1.4 FuncTion	15
7.1.5 MemCAD	15
7.1.6 KAPPA	16
7.1.7 PYPPAI	16
7.1.8 Libra	16
7.1.9 Lyra	17
7.1.10 Typpete	17
7.1.11 ApronPy	17
8 New results	17
8.1 Symbolic Numerical Domains	17
8.2 Security Properties	18
8.3 Quantitative Properties	18
8.4 Functional properties, hyperproperties, and quantitative properties	19
8.5 Principles of static analysis	19

8.6	Static analysis applied to the verification of operating systems	19
8.7	Imprecision in static analysis	20
8.8	Static Analysis of Probabilistic Programming Languages and Optimization Algorithms	20
8.9	Data Science and Machine Learning	20
8.10	Concurrency	23
8.11	Application to Distributed Applications	23
8.12	Application to Computational Systems Biology	24
9	Partnerships and cooperations	25
9.1	International initiatives	25
9.1.1	Inria associate team not involved in an IIL or an international program	25
9.2	National initiatives	25
9.2.1	ANR EMASS	25
9.2.2	ANR ForML	26
9.2.3	PEPR SecurEval	26
9.2.4	PEPR SAIF	27
10	Dissemination	27
10.1	Promoting scientific activities	27
10.1.1	Scientific events: organisation	27
10.1.2	Scientific events: selection	28
10.1.3	Journal	29
10.1.4	Invited talks	29
10.1.5	Leadership within the scientific community	29
10.1.6	Scientific expertise	29
10.1.7	Recruiting Juries	30
10.1.8	Research administration	30
10.2	Teaching - Supervision - Juries - Educational and pedagogical outreach	30
10.2.1	Teaching	30
10.2.2	Entrance competition juries	31
10.2.3	Supervision	31
10.2.4	PhD Juries	31
10.3	Popularization	32
10.3.1	Specific official responsibilities in science outreach structures	32
11	Scientific production	32
11.1	Major publications	32
11.2	Publications of the year	33
11.3	Cited publications	35

1 Team members, visitors, external collaborators

Research Scientists

- Jerome Feret [Team leader, INRIA, Researcher, until Sep 2025, HDR]
- Jerome Feret [Team leader, INRIA, Senior Researcher, from Oct 2025, HDR]
- Bernadette Charron-Bost [LIX, HDR]
- Vincent Danos [CNRS, Senior Researcher, HDR]
- Hugo Paquet [INRIA, Researcher]
- Xavier Rival [INRIA, Senior Researcher, HDR]
- Caterina Urban [INRIA, Researcher, HDR]

Post-Doctoral Fellows

- Marco Campion [INRIA, Post-Doctoral Fellow, until Aug 2025]
- Alessandro De Palma [INRIA, Post-Doctoral Fellow, until Oct 2025]
- Guannan Wei [INRIA, Post-Doctoral Fellow, until Jul 2025]

PhD Students

- Valentin Barbazo [ENS Paris]
- Jerome Boillot [INRIA, from Sep 2025]
- Jerome Boillot [ENS PARIS, until Aug 2025]
- Yoan Bouniard [INRIA, from Sep 2025]
- Charles De Haro [ENS Paris]
- Serge Durand [INRIA, until Oct 2025]
- Rebecca Ghidini [CNRS, from Sep 2025]
- Serge Lechenne [ENS, from Sep 2025]
- Priyanka Maity [INRIA, from Nov 2025]
- Naim Moussaoui Remil [INRIA]
- Patricia Tenera Roxo [ENS Paris]

Technical Staff

- Yoan Bouniard [INRIA, Engineer, until Feb 2025]
- Rebecca Ghidini [CNRS, until Aug 2025, Engineer]
- Heyuan Liu [INRIA, Engineer, from Dec 2025]
- Priyanka Maity [INRIA, from Oct 2025 until Oct 2025]
- Antoine Pouille [INRIA, Engineer, until May 2025]

Interns and Apprentices

- Yoan Bouniard [INRIA, from Mar 2025 until Aug 2025]
- Sarah Dribi Alaoui [INRIA, Intern, from Jun 2025 until Sep 2025]
- Serge Lechenne [ENS PARIS, Intern, from Mar 2025 until Aug 2025]
- Thomas Winninger [INRIA, Intern, from Mar 2025 until May 2025]

Administrative Assistants

- Meriem Guemair [INRIA]
- Diana Marino Duarte [INRIA]
- Abigail Palma [INRIA]

External Collaborators

- Greta Dolcetti [Univ Ca' Foscari, from Feb 2025]
- Lorenzo Gazzella [University of Pisa, from Sep 2025]
- Giacomo Zanatta [Univ Ca' Foscari, from Feb 2025]

2 Overall objectives

Our group focuses on developing *automated* techniques to compute *semantic properties* of programs and other systems with a computational semantics in general. Such properties include (but are not limited to) important classes of correctness properties.

2.1 Verifying safety critical systems

Verifying safety critical systems (such as avionics systems) is an important motivation to compute such properties. Indeed, a fault in an avionics system, such as a runtime error in the fly-by-wire command software, may cause an accident, with loss of lives. As these systems are also very complex and are developed by large teams and maintained over long periods, their verification has become a crucial challenge. Safety critical systems are not limited to avionics: software runtime errors in cruise control management systems were also blamed for causing *unintended acceleration* in certain Toyota models (the case was settled with a 1.2 billion dollars fine in March 2014, after years of investigation and several trials). Similarly, other transportation systems (railway), energy production systems (nuclear power plants, power grid management), medical systems (pacemakers, surgery and patient monitoring systems), and value transfers in decentralized systems (smart contracts), rely on complex software, which should be verified.

Beyond the field of embedded systems, other pieces of software may cause very significant harm in the case of bugs, as demonstrated by the Heartbleed security hole: due to a wrong protocol implementation, many websites could leak private information, over years.

2.2 Formal definition of safety properties

An important example of semantic properties is the class of *safety* properties. A safety property typically specifies that some (undesirable) event will never occur, whatever the execution of the program that is considered. For instance, the absence of runtime error is a very important safety property. Other important classes of semantic properties include *liveness* properties (i.e., properties that specify that some desirable event will eventually occur) such as termination and *security* properties, such as the absence of information flows from private to public channels. All these software semantic properties are *not decidable*, as can be shown by reduction to the halting problem. Therefore, there is no chance to develop any fully automatic technique able to decide, for any system, whether or not it satisfies some given semantic property.

2.3 Other verification approaches

The classic development techniques used in industry involve testing, which is not sound, as it only gives information about a usually limited test sample: even after successful test-based validation, situations that were untested may generate a problem. Furthermore, testing is costly in the long term, as it should be re-done whenever the system to verify is modified. Machine-assisted verification is another approach which verifies human specified properties. However, this approach also presents a very significant cost, as the annotations required to verify large industrial applications would be huge.

By contrast, the **Antique** group focuses on the design of semantic analysis techniques that should be *sound* (i.e., compute semantic properties that are satisfied by all executions) and *automatic* (i.e., with no human interaction), although generally *incomplete* (i.e., not able to compute the best—in the sense of: most precise— semantic property). As a consequence of incompleteness, we may fail to verify a system that is actually correct. For instance, in the case of verification of absence of runtime error, the analysis may fail to validate a program, which is safe, and emit *false alarms* (that is reports that possibly dangerous operations were not proved safe), which need to be discharged manually. Even in this case, the analysis provides information about the alarm context, which may help disprove it manually or refine the analysis.

2.4 Emergent applications

The methods developed by the **Antique** group are not limited to the analysis of software.

We also consider complex biological systems (such as models of signaling pathways, i.e. cascades of protein interactions, which enable signal communication among and within cells), described in higher level languages, and use abstraction techniques to reduce their combinatorial complexity and capture key properties to get a better insight in the underlying mechanisms of these systems.

The **Antique** group is also applying its approaches to the software that are developed in the context of data science and machine learning to prove their robustness and automatically derive explanations.

3 Research program

3.1 Semantics

Understanding the formal semantics of a programming language is the first step in any program analysis. Our group works to extend the reach of semantic methods to new or non-standard kinds of programming languages. We also work on the mathematical foundations of program semantics, to clarify the concepts and methods. This foundational research is always carried out with a motivation in practical static analysis.

3.1.1 The role of semantics in program verification

Semantics plays a central role in verification since it always serves as a basis to express the properties of interest, that need to be verified, but also additional properties, required to prove the properties of interest, or which may make the design of static analysis easier.

For instance, if we aim for a static analysis that should prove the absence of runtime error in some class of programs, the concrete semantics should define properly what error states and non error states are, and how program executions step from a state to the next one. In the case of a language like C, this includes the behavior of floating point operations as defined in the IEEE 754 standard. When considering parallel programs, this includes a model of the scheduler, and a formalization of the memory model.

Additionally, it is sometimes desirable for the semantics to describe program behavior with more precision than is strictly necessary for the property of interest. For instance, to establish a state property (such as the absence of runtime errors) we need a state invariant, but the analysis is sometimes significantly easier using a *trace* invariant, which also tracks the order in which states are visited, and allows for case-splitting on possible execution paths. This is only possible if the semantic model describes not just program states but also program traces.

Thus, for a given programming language there may be a number of useful semantics, tracking different aspects of program execution. One challenge is to understand how different semantics relate to each other, so that benefits can be compared or combined.

3.1.2 Understanding program semantics using category theory

On the theoretical side, our group studies program semantics using the abstract language of category theory. The benefit of working at a higher level of abstraction is that it is easier to develop general methods, applicable to all kinds of programming languages.

The motivating observation for this line of work is that a mathematical model for a programming language can easily be organized as a category. This presentation emphasizes the compositional aspects of the language. Category theory also makes it easier to relate different semantic models for the same language, since they are all formalized in a unified setting.

Category theory is also a convenient mathematical language for reasoning about side effects and type systems. As an example, one useful categorical abstraction is the notion of monad, which encapsulates a particular instance of side effect. Monads have a very well-understood categorical theory and are also used in practice. For instance there are monads for state effects (used for imperative programming) and for probabilistic effects (used for probabilistic programming).

Category theory is also broadly used to formalize the semantics of rewriting languages (as those used for modelling biological systems).

3.2 Abstract interpretation and static analysis

Once a reference semantics has been fixed and a property of interest has been formalized, the definition of a static analysis requires the choice of an *abstraction*. The abstraction ties a set of *abstract predicates* to the concrete ones, which they denote. This relation is often expressed with a *concretization function* that maps each abstract element to the concrete property it stands for. Obviously, a well-chosen abstraction should allow one to express the property of interest, as well as all the intermediate properties that are required in order to prove it (otherwise, the analysis would have no chance to achieve a successful verification). It should also lend itself to an efficient implementation, with efficient data-structures and algorithms for the representation and the manipulation of abstract predicates. A great number of abstractions have been proposed for all kinds of concrete data types, yet the search for new abstractions is a very important topic in static analysis, to target novel kinds of properties, to design more efficient or more precise static analyses.

Once an abstraction is chosen, a set of *sound abstract transformers* can be derived from the concrete semantics and that account for individual program steps, in the abstract level and without forgetting any concrete behavior. A static analysis follows as a result of this step by step approximation of the concrete semantics, when the abstract transformers are all computable. This process defines an *abstract interpretation* [33]. The case of loops requires a bit more work as the concrete semantics typically relies on a fix point that may not be computable in finitely many iterations. To achieve a terminating analysis we then use *widening operators* [33], which over-approximate the concrete union and ensure termination.

A static analysis defined that way always terminates and produces sound over-approximations of the programs behaviors. Yet, these results may not be precise enough for verification. This is where the art of static analysis design comes into play through, among others:

- the use of more precise, yet still efficient enough abstract domains;
- the combination of application-specific abstract domains;
- the careful choice of abstract transformers and widening operators.

3.3 Applications of the notion of abstraction in semantics

In the previous subsections, we sketched the steps in the design of a static analyzer to infer some families of properties, which should be implementable, and efficient enough to succeed in verifying non-trivial systems.

The same principles can be applied successfully to other goals. In particular, the abstract interpretation framework should be viewed as a very general tool to *compare different semantics*, not necessarily with the goal of deriving a static analyzer. Such comparisons may be used in order to prove two semantics equivalent (i.e., one is an abstraction of the other and vice versa), or that a first semantics is strictly more expressive than another one (i.e., the latter can be viewed an abstraction of the former, where the abstraction actually makes some information redundant, which cannot be recovered). A classical example of such comparison is

the classification of semantics of transition systems [32], which provides a better understanding of program semantics in general. For instance, this approach can be applied to get a better understanding of the semantics of a programming language, but also to select which concrete semantics should be used as a foundation for a static analysis, or to prove the correctness of a program transformation, compilation or optimization.

3.4 From properties to explanations

In many application domains, we can go beyond the proof that a program satisfies its specification. Abstractions can also offer new perspectives to understand how complex behaviors of programs emerge from simpler computation steps. Abstractions can be used to find compact and readable representations of sets of traces, causal relations, and even proofs. For instance, abstractions may decipher how the collective behaviors of agents emerge from the orchestration of their individual ones in distributed systems (such as consensus protocols, models of signaling pathways). Another application is the assistance for the diagnostic of alarms of a static analyzer.

Often times, complex systems and software have intricate behaviors, leading to executions that are hard to understand for programmers and also difficult to reason about with static analyzers. An additional challenge arises in the distributed setting with the combinatorial explosion of the number of possible executions inherent to the non-determinism of the system (interleaving of actions performed by different processes and link or process failures). In 2009, Charron-Bost and Schiper [5] proposed a new computing model, namely the Heard-Of model, in which computations evolve in synchronous rounds that are communication-closed layers. At each round r , information transmissions are represented by the communication graph, and each execution corresponds to a sequence of communication graphs, i.e., a dynamic graph. The features of a specific system are thus captured as a whole, just by the collection of possible dynamic graphs. This model handles various asynchronicity degrees and benign failures, be they static or dynamic, permanent or transient, in a unified framework: synchrony assumptions and failure model in a static network are translated into a certain class of dynamic graphs. Due to its high abstraction level and its expressiveness, the Heard-Of model has gained a considerable attention of the verification community, in particular for the *consensus* algorithms. Subsequently, this model was successfully used to study other fundamental problems in distributed computing such as *synchronization* and *leader election*.

Process crashes play a key role in the behavior of distributed systems. In Erlang, for instance, process crashes involve cascades of signals which aims at recovering state consistency. Yet, the order of signal handling is a potential source of bugs, the precise origin of which can be difficult to understand. Addressing this issue requires a low-level semantics describing precisely signal handling (as the one we proposed in [36]) and causal analysis to understand why a system deviate from the behaviors in its specification (as the one that is implanted in CESAn [16, 29], e.g. see Sect. 7.1.3).

In models of signaling pathways, collective behavior emerges from competition for common resources, separation of scales (time/concentration), non-linear feedback loops, which are all consequences of mechanistic interactions between individual bio-molecules (e.g., proteins). While more and more details about mechanistic interactions are available in the literature, understanding the behavior of these models at the system level is far from easy. Causal analysis helps explain how specific events of interest may occur. Model reduction techniques combine methods from different domains such as the analysis of information flow used in communication protocols, and tropicalization methods that comes from physics. The result is lower dimension systems that preserve the behavior of the initial system while focusing of the elements from which emerges the collective behavior of the system.

The abstraction of causal traces offers nice representation of scenarios that lead to expected or unexpected events. This is useful to understand the necessary steps in potential scenarios in signaling pathways; this is useful as well to understand the different steps of an intrusion in a protocol. Lastly, traces of computation of a static analyzer can themselves be abstracted, which provides assistance to classify true and false alarms. Abstracted traces are symbolic and compact representations of sets of counter-examples to the specification of a system which help one to either understand the origin of bugs, or to find that some information has been lost in the abstraction leading to false alarms.

4 Application domains

4.1 Verification of safety critical embedded software

The verification of safety critical embedded software is a major application domain for our group. Firstly, this field requires a high confidence in software, as a bug may cause disastrous events. Thus, it offers an obvious opportunity for a strong impact. Secondly, such software usually have better specifications and a better design than many other families of software, hence are an easier target for developing new static analysis techniques (which can later be extended for more general, harder to cope with families of programs). This includes avionics, automotive and other transportation systems, medical systems . . .

For instance, the verification of avionics systems represents a very high percentage of the cost of an airplane (about 30 % of the overall airplane design cost). The state-of-the-art development processes mainly resorts to testing in order to improve the quality of software. Depending on the level of criticality of a software (at the highest levels, any software failure would endanger the flight) a set of software requirements are checked with test suites. This approach is both costly (due to the sheer amount of testing that needs to be performed) and unsound (as errors may go unnoticed, if they do not arise on the test suite).

By contrast, static analysis can ensure higher software quality at a lower cost. Indeed, a static analyzer will catch all bugs of a certain kind. Moreover, a static analysis run typically lasts a few hours, and can be integrated in the development cycle in a seamless manner. For instance, **ASTRÉE** successfully verified the absence of runtime error in several families of safety critical fly-by-wire avionic software, in at most a day of computation, on standard hardware. Other kinds of synchronous embedded software have also been analyzed with good results. In the future, we plan to greatly extend this work to verify *other families of embedded software* (such as communication, navigation and monitoring software) and *other families of properties* (such as security and liveness properties).

Embedded software in charge of communication, navigation, and monitoring typically relies on a *parallel* structure, where several threads are executed concurrently, and manage different features (input, output, user interface, internal computation, logging . . .). This structure is also often found in automotive software. An even more complex case is that of *distributed* systems, where several separated computers are run in parallel and take care of several sub-tasks of a same feature, such as braking. Such a logical structure is not only more complex than the synchronous one, but it also introduces new risks and new families of errors (deadlocks, data-races...). Moreover, such less well-designed, and more complex embedded software often utilizes more complex data-structures than synchronous programs (which typically only use arrays to store previous states) and may use dynamic memory allocation, or build dynamic structures inside static memory regions, which are actually even harder to verify than conventional dynamically allocated data structures. Complex data-structures also introduce new kinds of risks (the failure to maintain structural invariants may lead to runtime errors, non-termination, or other software failures). To verify such programs, we will design additional abstract domains, and develop new static analysis techniques. Hence, we will support the analysis of parallel and concurrent programming with threads and manipulations of complex data structures. Due to their size and complexity, the verification of such families of embedded software is a major challenge for the research community.

Furthermore, embedded systems also give rise to novel security concerns. It is in particular the case for some aircraft-embedded computer systems, which communicate with the ground through untrusted communication media. Besides, the increasing demand for new capabilities, such as enhanced on-board connectivity, e.g. using mobile devices, together with the need for cost reduction, leads to more integrated and interconnected systems. For instance, modern aircrafts embed a large number of computer systems, from safety-critical cockpit avionics to passenger entertainment. Some systems meet both safety and security requirements. Despite thorough segregation of subsystems and networks, some shared communication resources raise the concern of possible intrusions. Because of the size of such systems, and considering that they are evolving entities, the only economically viable alternative is to perform automatic analyses. Such analyses of security and confidentiality properties have never been achieved on large-scale systems where security properties interact with other software properties, and even the mapping between high-level models of the systems and the large software base implementing them has never been done and represents a great challenge. Our goal is to prove empirically that the security of such large scale systems can be proved formally, thanks to the design of dedicated abstract interpreters.

The long term goal is to make static analysis more widely applicable to the verification of industrial

software.

4.2 Static analysis of software components and libraries

An important goal of our work is to make static analysis techniques easier to apply to wider families of software. Then, in the longer term, we hope to be able to verify less critical, yet very commonly used pieces of software. Those are typically harder to analyze than critical software, as their development process tends to be less rigorous. In particular, we will target operating system components and libraries. As of today, the verification of such programs is considered a major challenge to the static analysis community.

As an example, most programming languages offer Application Programming Interfaces (API) providing ready-to-use abstract data structures (e.g., sets, maps, stacks, queues, etc.). These APIs, are known under the name of containers or collections, and provide off-the-shelf libraries of high level operations, such as insertion, deletion and membership checks. These container libraries give software developers a way of abstracting from low-level implementation details related to memory management, such as dynamic allocation, deletion and pointer handling or concurrency aspects, such as thread synchronization. Libraries implementing data structures are important building bricks of a huge number of applications, therefore their verification is paramount. We are interested in developing static analysis techniques that will prove automatically the correctness of large audience libraries such as Glib and Threading Building Blocks.

4.3 Models of mechanistic interactions between proteins

Computer Science takes a more and more important role in the design and the understanding of biological systems such as signaling pathways, self assembly systems, DNA repair mechanisms. Biology has gathered large data-bases of facts about mechanistic interactions between proteins, but struggles to draw an overall picture of how these systems work as a whole. High level languages designed in Computer Science allow one to collect these interactions in integrative models, and provide formal definitions (i.e., semantics) for the behavior of these models. This way, modelers can encode their knowledge, following a bottom-up discipline, without simplifying *a priori* the models at the risk of damaging the key properties of the system. Yet, the systems that are obtained this way suffer from combinatorial explosion (in particular, in the number of different kinds of molecular components, which can arise at run-time), which prevents from a naive computation of their behavior.

We develop various analyses based on abstract interpretation, and tailored to different phases of the modeling process. We propose automatic static analyses in order to detect inconsistencies in the early phases of the modeling process. These analyses are similar to the analysis of classical safety properties of programs. They involve both forward and backward reachability analyses as well as causality analyses, and can be tuned at different levels of abstraction. We also develop automatic static analyses in order to identify key elements in the dynamics of these models. The results of these analyses are sent to another tool, which is used to automatically simplify models. The correctness of this simplification process is proved by the means of abstract interpretation: this ensures formally that the simplification preserves the quantitative properties that have been specified beforehand by the modeler. The whole pipeline is parameterized by a large choice of abstract domains which exploits different features of the high level description of models.

4.4 Application of distributed algorithms

We mainly worked on two different topics, namely synchronization in multi-agent networks and constrained consensus (or equivalently, function computation). We studied the two problems in the round-based Heard-Of model (cf. Section 3.4), where each computation is determined by a *dynamic graph* formed by the sequence of communication graphs at each round. As above explained, the features of the network (synchrony degree, failures, knowledge, ...) are captured by the class of possible dynamic graphs.

Synchronization in multi-agent networks. We consider the fundamental problem of periodic clock synchronization in a synchronous multi-agent system: Each agent holds a clock with an arbitrary initial value, and clocks must eventually be congruent, modulo some positive integer P . Our main contribution has been to prove some lower bounds on synchronization time and memory size required at each agent to synchronize, and to propose a new synchronization algorithm, called *SAP* because it is based on a *self-adaptive period*

mechanism, which fits these lower bounds. Interestingly, our algorithm works under very weak connectivity assumptions, and do not require any global knowledge on the network. Its correctness has been established in the proof assistant Isabelle. Then we extended these results to probabilistic communication networks: we demonstrated that the SAP algorithm still works in any probabilistic communication network that is "connected with high probability". The proof of such a probabilistic *hyperproperty* is based on novel tools and relies on weak assumptions about the probabilistic communication network, making it applicable to a wide range of networks, including the classical rumor spreading models.

Constrained consensus. The problem consists, for a networked system of autonomous agents, to collectively compute the value of a given function of some input values, each initially private to one agent in the network. In the "blind broadcast" model, where an agent merely casts out a unique message without any knowledge or control over its addressees, the computable functions are exactly those that only depend on the *set* of the input values. In contrast, we proved that, when either i) the agents know how many outneighbors they have; ii) all communications links in the network are bidirectional; or iii) the agents may address each of their outneighbors individually, the set of computable functions grows to contain all functions that depend on the relative *frequencies* of each value in the input such as the average. We first obtained all these exact characterizations of computable functions for static networks using the notion of graph fibration in homotopy graph theory. The same characterizations hold for dynamic networks. To show this, we developed a totally different method based on the stochastic analysis of consensus algorithms derived from statistical physics.

4.5 Formal reasoning about parallel program execution

We are interested in developing new mathematical tools for reasoning about the semantics of parallel program execution. It is well-known that parallel execution is difficult to reason about for imperative programs, because of potential data races that induce non-determinism. In this line of research, we exploit modern tools in semantics (monads and 2-dimensional categories) to obtain a better grasp of the semantic issues induced by parallel execution.

The methods we develop are applicable beyond simple imperative programs. Indeed, parallel execution is a semantic challenge for programming with mutable state, but also for many other forms of programming, including probabilistic programming and programming with first-class continuations, for which we have developed specialized abstractions.

The extensive literature on mathematical models for concurrency (Petri nets, event structures, asynchronous transition systems, *etc.*) plays an important role in this work, because these structures provide a precise mathematical language for causality and races. One objective is to support the development of program logics for concurrency—an active research area in programming languages.

4.6 Static analysis of data science and machine learning software

Nowadays, thanks to advances in machine learning and the availability of vast amounts of data, computer software plays an increasingly important role in assisting or even autonomously performing tasks in our daily lives. As data science software becomes more and more widespread, we become increasingly vulnerable to programming errors. In particular, programming errors that do not cause failures can have serious consequences since code that produces an erroneous but plausible result gives no indication that something went wrong. This issue becomes particularly worrying knowing that machine learning software, thanks to its ability to efficiently approximate or simulate more complex systems, is slowly creeping into mission critical scenarios. However, programming errors are not the only concern. Another important issue is the vulnerability of machine learning models to adversarial examples, that is, small input perturbations that cause the model to misbehave in unpredictable ways. More generally, a critical issue is the notorious difficulty to interpret and explain machine learning software. Finally, as we are witnessing widespread adoption of software with far-reaching societal impact — i.e., to automate decision-making in fields such as social welfare, criminal justice, and even health care — a number of recent cases have evidenced the importance of ensuring software fairness as well as data privacy. Going forward, data science software will be subject to more and more legal regulations (e.g., the European General Data Protection Regulation adopted in 2016) as well as administrative audits.

It is thus paramount to develop method and tools that can keep up with these developments and enhance our understanding of data science software and ensure it behaves correctly and reliably. In particular, we are interesting in developing new static analyses specifically tailored to the idiosyncrasies of data science software. This makes it a new and exciting area for static analysis, offering a wide variety of challenging problems with huge potential impact on various interdisciplinary application domains [37].

4.7 Probabilistic programming and statistical inference

Probabilistic programming is a method for Bayesian probabilistic modelling in which statistical models are encoded as programs. Compared with the traditional approach, in which models are written down in mathematical notation, the advantage of the programming environment is that complex models can be clearly structured, and users can easily experiment with different models. Additionally, a probabilistic programming language typically comes with various built-in tools for *inference*—the process of updating probability distributions in the model after observing part of the data. This is a notoriously difficult practical problem and the central goal of Bayesian statistics.

Our research is mainly concerned with ensuring that inference algorithms from traditional statistics can be correctly adapted to probabilistic programming, in the sense that, for any probabilistic program, the results of inference are mathematically consistent with statistical theory.

In some cases, standard results from probability theory can be leveraged for correctness, but for advanced inference algorithms this usually requires care. For example, variational inference methods are only correct under additional assumptions, and we develop static analyses to verify these assumptions automatically for probabilistic programs. Another example is the correctness of an optimized Metropolis-Hastings algorithm. This is a very generic inference algorithm, which works for all programs, but generally performs poorly unless the program’s data dependencies are properly understood and exploited. We gave a formal semantic account of these dependencies to formally prove the optimization correct.

In addition to correctness results, we are interested in the expressive power of probabilistic programming for certain classes of infinite-dimensional models. Compositional semantics, including category-theoretic methods, provide a bridge between statistical models and probabilistic programs, which we develop in order to address these questions.

5 Social and environmental responsibility

5.1 Impact of research results

We are advising static analysis companies including AbsInt Angewandte Informatik (static analysis for the verification of embedded software) and MatrixLead (static analysis for spreadsheet applications).

6 Highlights of the year

6.1 Awards

- Marco Campion received the Radhia Cousot award at [SAS 2025 \(Static analysis symposium\)](#) for the paper [Relating Distances and Abstractions: An Abstract Interpretation Perspective](#) [19] cowritten by Marco Campion, Isabella Mastroeni, and Caterina Urban;
- Patrícia Tenera Roxo received the best student paper award at [CMSB 2025 \(Computational Methods in Systems Biology\)](#) for the article [Computation of Immediate Neighbors of Monotone Boolean Functions](#) [34] cowritten by José E.R. Cury, Patrícia Tenera Roxo, Vasco Manguinho, Claudine Chaouiya, and Pedro T. Monteiro.

6.2 Conference organisations

- Caterina Urban was the general chair of [iFM 2025 \(The International Conference on integrated Formal Methods\)](#) at Inria Paris Research Center.

7 Latest software developments, platforms, open data

7.1 Latest software developments

7.1.1 Astrée

Name: The AstréeA Static Analyzer of Asynchronous Software

Keywords: Static analysis, Static program analysis, Program verification, Software Verification, Abstraction

Scientific Description: Astrée analyzes structured C programs, with complex memory usages, but without dynamic memory allocation nor recursion. This encompasses many embedded programs as found in earth transportation, nuclear energy, medical instrumentation, and aerospace applications, in particular synchronous control/command. The whole analysis process is entirely automatic.

Functional Description: Astrée discovers all runtime errors including: - undefined behaviors in the terms of the ANSI C99 norm of the C language (such as division by 0 or out of bounds array indexing), - any violation of the implementation-specific behavior as defined in the relevant Application Binary Interface (such as the size of integers and arithmetic overflows), - any potentially harmful or incorrect use of C violating optional user-defined programming guidelines (such as no modular arithmetic for integers, even though this might be the hardware choice), - failure of user-defined assertions.

Astrée is a static analyzer for sequential programs based on abstract interpretation. The Astrée static analyzer aims at proving the absence of runtime errors in programs written in the C programming language.

URL: <http://www.astree.ens.fr/>

Contact: Patrick Cousot

Participants: Antoine Mine, Jerome Feret, Laurent Mauborgne, Patrick Cousot, Radhia Cousot, Xavier Rival

Partners: CNRS, ENS Paris, AbsInt Angewandte Informatik GmbH

7.1.2 AstréeA

Name: The AstréeA Static Analyzer of Asynchronous Software

Keywords: Static analysis, Static program analysis

Scientific Description: AstréeA analyzes C programs composed of a fixed set of threads that communicate through a shared memory and synchronization primitives (mutexes, FIFOs, blackboards, etc.), but without recursion nor dynamic creation of memory, threads nor synchronization objects. AstréeA assumes a real-time scheduler, where thread scheduling strictly obeys the fixed priority of threads. Our model follows the ARINC 653 OS specification used in embedded industrial aeronautic software. Additionally, AstréeA employs a weakly-consistent memory semantics to model memory accesses not protected by a mutex, in order to take into account soundly hardware and compiler-level program transformations (such as optimizations). AstréeA checks for the same run-time errors as Astrée, with the addition of data-races.

Functional Description: AstréeA is a static analyzer prototype for parallel software based on abstract interpretation. The AstréeA prototype is a fork of the Astrée static analyzer that adds support for analyzing parallel embedded C software.

URL: <http://www.astree.ens.fr/>

Contact: Patrick Cousot

Participants: Antoine Mine, Jerome Feret, Patrick Cousot, Radhia Cousot, Xavier Rival

Partners: CNRS, ENS Paris, AbsInt Angewandte Informatik GmbH

7.1.3 CESAn

Name: Core Erlang Semantics Analyzer

Keyword: Formal semantics

Functional Description: CESAn implements the semantics of a subset of Core Erlang. Given a Core Erlang program with finite semantics, it outputs its semantics in the form of a labeled transition system. The underlying small-step semantics faithfully represents message passing and signal handling in Erlang.

Publications: [hal-04222884](#), [hal-05353119](#)

Contact: Aurélie Kong Win Chang

Participants: Aurélie Kong Win Chang, Jerome Feret, Gregor Goessler

7.1.4 FuncTion

Keywords: Static analysis, Termination

Scientific Description: FuncTion is based on an extension to liveness properties of the framework to analyze termination by abstract interpretation proposed by Patrick Cousot and Radhia Cousot. FuncTion infers ranking functions using piecewise-defined abstract domains. Several domains are available to partition the ranking function, including intervals, octagons, and polyhedra. Two domains are also available to represent the value of ranking functions: a domain of affine ranking functions, and a domain of ordinal-valued ranking functions (which allows handling programs with unbounded non-determinism).

Functional Description: FuncTion is a research prototype static analyzer to analyze the termination and functional liveness properties of programs. It accepts programs in a small non-deterministic imperative language. It is also parameterized by a property: either termination, or a recurrence or a guarantee property (according to the classification by Manna and Pnueli of program properties). It then performs a backward static analysis that automatically infers sufficient conditions at the beginning of the program so that all executions satisfying the conditions also satisfy the property.

URL: <http://www.di.ens.fr/~urban/FuncTion.html>

Contact: Caterina Urban

Participants: Antoine Mine, Caterina Urban

7.1.5 MemCAD

Name: The MemCAD static analyzer

Keywords: Static analysis, Abstraction

Functional Description: MemCAD is a static analyzer that focuses on memory abstraction. It takes as input C programs, and computes invariants on the data structures manipulated by the programs. It can also verify memory safety. It comprises several memory abstract domains, including a flat representation, and two graph abstractions with summaries based on inductive definitions of data-structures, such as lists and trees and several combination operators for memory abstract domains (hierarchical abstraction, reduced product). The purpose of this construction is to offer a great flexibility in the memory abstraction, so as to either make very efficient static analyses of relatively simple programs, or still quite efficient static analyses of very involved pieces of code. The implementation consists of over 30 000 lines of ML code, and relies on the ClangML front-end. The current implementation comes with over 300 small size test cases that are used as regression tests.

URL: <http://www.di.ens.fr/~rival/memcad.html>

Contact: Xavier Rival

Participants: Antoine Toubhans, Francois Berenger, Huisong Li, Xavier Rival

7.1.6 KAPPA

Name: A rule-based language for modeling interaction networks

Keywords: Systems Biology, Computational biology

Scientific Description: OpenKappa is a collection of tools to build, debug and run models of biological pathways. It contains a compiler for the Kappa Language, a static analyzer (for debugging models), a simulator, a compression tool for causal traces, and a model reduction tool.

Functional Description: Kappa is provided with the following tools: - a compiler - a stochastic simulator - a static analyzer - a trace compression algorithm - an ODE generator - a tool for parameter calibration

Release Contributions: Online user interface simulation is based on a new datastructure (see ESOP 2017), New abstract domains are available in the static analyzer (see SASB 2016), Local traces (see TCBB 2018), Reasoning on polymers (see SASB 2018). Furthermore, the ecosystem now includes a tool provided by Matthieu Bouguéon to calibrate the interaction rates in the underlying continuous time Markov chain.

URL: <http://www.kappalanguage.org/>

Contact: Jerome Feret

Participants: Rebecca Ghidini, Jean Krivine, Jerome Feret, Kim-Quyen Ly, Pierre Boutillier, Russ Harmer, Vincent Danos, Walter Fontana, Antoine Pouille, Matthieu Bougueon

7.1.7 PYPPAI

Name: Pyro Probabilistic Program Analyzer

Keywords: Probability, Static analysis, Program verification, Abstraction

Functional Description: PYPPAI is a program analyzer to verify the correct semantic definition of probabilistic programs written in Pyro. At the moment, PYPPAI verifies consistency conditions between models and guides used in probabilistic inference programs.

PYPPAI is written in OCaml and uses the pyml Python in OCaml library. It features a numerical abstract domain based on Apron, an abstract domain to represent zones in tensors, and dedicated abstract domains to describe distributions and states in probabilistic programs.

URL: <https://github.com/wonyeol/static-analysis-for-support-match>

Contact: Xavier Rival

Participant: Xavier Rival

7.1.8 Libra

Keywords: Fairness, Neural networks, Abstraction, Static analysis

Functional Description: Libra is an abstract interpretation-based static analyzer for certifying that ReLU neural network classifiers for tabular data are independent of their input values that are sensitive to bias. Libra combines a sound forward pre-analysis with an exact backward analysis that leverages the polyhedra abstract domain to provide definite fairness guarantees when possible, and to otherwise quantify and describe the biased input space regions. The analysis is configurable in terms of scalability

and precision. Libra is equipped with various abstract domains choices for the pre-analysis, including a generic reduced product domain construction, as well as search heuristics to find the best analysis configuration.

URL: <https://github.com/caterinaurban/Libra>

Contact: Caterina Urban

Participants: Caterina Urban, Denis Mazzucato

7.1.9 Lyra

Keyword: Static analysis

Functional Description: Lyra is a prototype static analyzer for data science applications written in Python.

URL: <https://github.com/caterinaurban/Lyra>

Contact: Caterina Urban

7.1.10 Typpete

Keywords: Static analysis, Satisfiability modulo theories

Functional Description: Typpete is a sound type inferencer that automatically infers Python 3 type annotations. Typpete encodes type constraints as a MaxSMT problem and uses optional constraints and specific quantifier instantiation patterns to make the constraint solving process efficient.

URL: <https://github.com/caterinaurban/Typpete>

Contact: Caterina Urban

7.1.11 ApronPy

Keywords: Static analysis, Abstract interpretation

Functional Description: Python Interface for the APRON Numerical Abstract Domain Library (<https://github.com/antoinemine/apron>).

URL: <https://github.com/caterinaurban/apronpy>

Contact: Caterina Urban

8 New results

8.1 Symbolic Numerical Domains

Abstraction of Memory Block Manipulations by Symbolic Loop Folding

Participants: Jérôme Boillot, Jérôme Feret.

In the paper [17], which has been accepted to ESOP 2025, we introduce a new abstract domain for analyzing memory block manipulations, focusing on programs with dynamically allocated arrays. This domain computes properties universally quantified over the value of the loop counters, both for assignments and tests. These properties consist of equalities and comparison predicates involving abstract expressions, represented as affine forms in the loop counters and symbolic dereferences. A well-founded order on the assignments prevents explosion of the abstract memory state.

All these methods have been incorporated within the *ASTRÉE* static analyzer that checks for the absence of run-time errors in embedded critical software. We also give insights on how to implement this abstract domain within any other C static analyzer.

8.2 Security Properties

Termination Resilience Static Analysis

Participants: Naïm Moussaoui Remil, Caterina Urban.

In [26], we present a novel abstract interpretation-based static analysis framework for proving Termination Resilience, the absence of Robust Non-Termination vulnerabilities in software systems. Robust Non-Termination characterizes programs where an untrusted (e.g., externally controlled) input can force infinite execution, independently of other trusted (e.g., controlled) variables. Our framework is a semantic generalization of Cousot and Cousot’s abstract interpretation-based ranking function derivation, and our sound static analysis extends Urban and Miné’s decision tree abstract domain in a non-trivial way to manage the distinction between untrusted and trusted program variables. Our approach is implemented in an open-source tool and evaluated on benchmarks sourced from SV-COMP and modeled after real-world software, demonstrating practical effectiveness in verifying Termination Resilience and detecting potential Robust Non-Termination vulnerabilities.

8.3 Quantitative Properties

Relating Distances and Abstractions: An Abstract Interpretation Perspective

Participants: Marco Campion, Isabella Mastroeni, Caterina Urban.

In [19], we establish a formal relation between quantitative and semantic approximations—formalized by pre-metrics and upper closure operators (ucos), respectively—by means of Galois connections. This connection reveals that it is far from trivial for a pre-metric to uniquely identify a uco, highlighting the structural constraints and, more generally, the distinct identity inherent to semantic approximations. Building on this foundation, we introduce a general composition of semantic and quantitative approximations. This allows us to define a new confidentiality property, called Partial Abstract Non-Interference, that measures bounded variations in program behavior over abstract properties of data. We then relate this property to Partial Completeness in abstract interpretation, revealing a deeper connection between static analysis precision and security guarantees.

This publication received the Radhia Cousot award at [SAS 2025 \(Static analysis symposium\)](#).

Abstract Lipschitz Continuity

Participants: Marco Campion, Isabella Mastroeni, Michele Pasqua, Caterina Urban.

In [31], we introduce Abstract Lipschitz Continuity (ALC), a generalization of standard Lipschitz Continuity, that ensures proportionally bounded differences in the semantic approximations of outputs when the semantic approximations of inputs differ slightly. ALC distinguishes between two complementary notions of approximation: quantitative differences, expressed via pre-metrics, and qualitative (or semantic) differences, captured through upper closure operators. ALC allows for reasoning about bounded changes in output properties in settings where standard Lipschitz continuity is too restrictive or inapplicable, such as in program analysis and verification, where understanding semantic properties of inputs and outputs is of key importance. In the specific context of programs, we formally relate ALC to other well-established program properties, including (Partial) Completeness and (Abstract) program Robustness. Notably, we show that ALC is a stronger requirement than Partial Completeness, a consolidated notion modeling precision loss

in program analysis. Finally, we propose a language- and domain-agnostic deductive system, parametric on the quantitative and semantic approximations of interest, for proving the ALC of programs. The goal in designing this deductive system is to track the assumptions required for ALC to ensure a compositional proof.

8.4 Functional properties, hyperproperties, and quantitative properties

Participants: Caterina Urban.

In her HDR thesis [30], Caterina Urban presents an overview of her research journey aimed at enhancing the quality and reliability of modern software systems through advanced static analyses. Rooted in the theory of abstract interpretation, her 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, her 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.

8.5 Principles of static analysis

Participants: Wonyeol Lee, Matthieu Lemerre, Xavier Rival, Hongseok Yang.

Static analysis aims at computing semantic properties of programs. Abstract interpretation provides a framework to design static analyses and allows one to divide the construction of a static analysis into the definition of abstract domains that describe families of logical predicates with operations to reason on them, and the semantic-guided formalization of abstract interpreters. The latter relies on the abstract domains, that describe semantic properties, and on the concrete semantics. A large part of the research on static analysis focuses on the design of novel abstract domains, with ever more expressive and/or efficient computer representation for semantic properties. In [25], we consider more specifically the core of the abstract interpreters (also called the abstract iterators) and discuss several techniques to build them, that are inspired by functional programming. First, we briefly discuss common iteration techniques based on control flow graphs, which have often been used for program analyzes aimed at computing state properties. Second, we consider iteration techniques that borrow principles from denotational semantics and are typically defined by induction over the syntax of programs. Last, we extend the latter family of techniques to relational abstractions that capture relations between program inputs and outputs.

8.6 Static analysis applied to the verification of operating systems

Verification of an instance of FreeRTOS

Participants: Josselin Giet, Julia Lawall, Gilles Muller, Nicolas Palix, Xavier Rival.

The functional correctness of operating kernels is notoriously very difficult to establish. We applied the MemCAD static analyzer to the verification of an instance of the FreeRTOS kernel. To achieve this result, we designed a specification language that allows to formalize the global invariant of the kernel, and the pre- and post-conditions of each system call into the kernel. For each such goal, MemCAD computes a sound abstract post-condition that over-approximates all executions starting from the specified pre-condition and attempts to discharge the implication of the specified post-condition by the abstract post-condition. By this technique, we could verify functional correctness of each system call.

8.7 Imprecision in static analysis

Logic for the Imprecision of Abstract Interpretations

Participants: Marco Campion, Mila Dalla Preda, Roberto Giacobazzi, Caterina Urban.

In numerical analysis, error propagation refers to how small inaccuracies in input data or intermediate computations accumulate and affect the final result, typically governed by the stability and sensitivity of the algorithm with respect to some perturbations. The definition of a similar concept in approximated program analysis is still a challenge. In abstract interpretation, inaccuracy arises from the abstraction itself, and the propagation of this error is dictated by the abstract interpreter. In most cases, such imprecision is inevitable.

In [11], we introduce a logic for deriving (upper) bounds on the inaccuracy of an abstract interpretation. We are able to derive a function that bounds the imprecision of the result of an abstract interpreter from the imprecision of its input data. When this holds we have what we call partial local completeness of the abstract interpreter, a weaker form of completeness known in the literature. To this end, we introduce the notion of a generator for a property represented in the abstract domain. Generators allow us to restrict the search space when verifying whether the bounding function holds for a given program and input. We then introduce a program logic, called Error Propagation Logic (EPL), for propagating the error bounds produced by an abstract interpretation. This logic is a combination of correctness and incorrectness logics and a logic for program ω -continuity that is also introduced in this paper.

8.8 Static Analysis of Probabilistic Programming Languages and Optimization Algorithms

Optimising Density Computations in Probabilistic Programs via Automatic Loop Vectorization

Participants: Hyoungjin Im, Wonyeol Lee, Sangho Lim, Xavier Rival, Hongseok Yang.

Probabilistic programming languages (PPLs) are a popular tool for high-level modelling across many fields. They provide a range of algorithms for probabilistic inference, which analyze models by learning their parameters from a dataset or estimating their posterior distributions. However, probabilistic inference is known to be very costly. One of the bottlenecks of probabilistic inference stems from the iteration over entries of a large dataset or a long series of random samples. Vectorization can mitigate this cost, but manual vectorization is error-prone, and existing automatic techniques are often ad-hoc and limited, unable to handle general repetition structures, such as nested loops and loops with data-dependent control flow, without significant user intervention.

To address this bottleneck, we propose in [14], a sound and effective method for automatically vectorizing loops in probabilistic programs. Our method achieves high throughput using speculative parallel execution of loop iterations, while preserving the semantics of the original loop through a fixed-point check. We formalize our method as a translation from an imperative PPL into a lower-level target language with primitives geared towards vectorization. We implemented our method for the Pyro PPL and evaluated it on a range of probabilistic models. Our experiments show significant performance gains against an existing vectorization baseline, achieving 1.1–6 \times speedups and reducing GPU memory usage in many cases. Unlike the baseline, which is limited to a subset of models, our method effectively handled all the tested models.

8.9 Data Science and Machine Learning

Static Analysis by Abstract Interpretation Against Data Leakage in Machine Learning Science of Computer

Participants: Caterina Urban, Pavle Subotić, Filip Drobnjaković.

Data leakage is a well-known problem in machine learning which occurs when the training and testing datasets are not independent. This phenomenon leads to unreliably overly optimistic accuracy estimates at training time, followed by a significant drop in performance when models are deployed in the real world. This can be dangerous, notably when models are used for risk prediction in high-stakes applications. In [15], we propose an abstract interpretation-based static analysis to prove the absence of data leakage at development time, long before model deployment and even before model training. We implemented it in the NBLyzer framework, and we demonstrate its performance and precision on 2111 Jupyter notebooks from the Kaggle competition platform.

This paper is the journal version of [35].

Over-Approximating Neural Networks for Verification, Robustness, and Explainability

Participants: Serge Durand, Zakaria Chihani, François Terrier, Caterina Urban.

The success of deep learning since the early 2010s has led to its adoption in a wide range of applications, from image classification to natural language processing. However, deploying AI models — especially in safety-critical applications — requires strong behavioral guarantees as well as means to understand and explain their predictions. Robustness — the ability of a model to maintain performance under small input changes — is an important safety requirement and helps in formally explaining model predictions. Over-approximation techniques, introduced by the machine learning and formal methods communities, have emerged as a key tool to verify neural networks and to train them with provable robustness goals. By bounding the range of possible network outputs, they make it possible to prove desirable properties for all possible inputs within a specified input domain.

In his PhD thesis [28], Serge Durand investigates how over-approximations can be effectively exploited to further improve the verification, training, and formal explainability of neural networks. Such over-approximations are essential to make the analysis tractable for neural networks of moderate scale, but they also tend to be imprecise, in particular for networks trained with no robustness goals. In a first contribution, key information from over-approximations is leveraged to efficiently tighten bounds when verifying trained networks with low-dimensional input spaces. This technique is implemented in PyRAT, a neural network verification tool, and contributed to its success in several benchmarks of the latest edition of the International Verification of Neural Networks Competition (VNN-COMP 2024). A second contribution shows that certified training - using over-approximations to minimize a sound bound on the worst-case loss - can be mixed with adversarial training - computing the loss over perturbed training samples - to improve empirical robustness and prevent catastrophic overfitting, a failure mode of single-step adversarial training, under specific experimental settings. Finally, it is proposed to use over-approximations to train for formal explainability. To avoid the trivial case where robustness voids all explanations, the notion of Feature Subset Certified Training is introduced. This new scheme enforces robustness only over selected subsets of input features, a first step towards better trade-offs between accuracy and conciseness of the explanations. Together, these contributions illustrate how over-approximations can be leveraged towards better robustness and explainability, supporting the development of safer and more trustworthy artificial intelligence systems.

On Using Certified Training towards Empirical Robustness

Participants: Alessandro De Palma, Serge Durand, Zakaria Chihani, Caterina Urban.

Adversarial training is arguably the most popular way to provide empirical robustness against specific adversarial examples. While variants based on multistep attacks incur significant computational overhead, single-step variants are vulnerable to a failure mode known as catastrophic overfitting, which hinders their practical utility for large perturbations. A parallel line of work, certified training, has focused on producing networks amenable to formal guarantees of robustness against any possible attack. However, the wide gap between the best-performing empirical and certified defenses has severely limited the applicability of the

latter. Inspired by recent developments in certified training, which rely on a combination of adversarial attacks with network over-approximations, and by the connections between local linearity and catastrophic overfitting, we present, in [13], experimental evidence on the practical utility and limitations of using certified training towards empirical robustness. We show that, when tuned for the purpose, a recent certified training algorithm can prevent catastrophic overfitting on single-step attacks, and that it can bridge the gap to multistep baselines under appropriate experimental settings. Finally, we present a conceptually simple regularizer for network over-approximations that can achieve similar effects while markedly reducing runtime.

Robustness to Perturbations in the Frequency Domain: Neural Network Verification and Certified Training

Participants: Harleen Hanspal, Alessandro De Palma, Alessio Lomuscio.

Deploying neural networks in safety critical applications such as autonomous driving requires assurance on their robustness. Deterministic robustness assessment can be made using formal verification. Existing frameworks for network verification and certified training verify and robustify networks against specifications capturing specific transformations or perturbations in the pixel or latent space. However, recent works highlight the vulnerability of networks to perturbations and attacks in the frequency domain which cannot be precisely captured by the existing specifications.

Therefore, we present in [24], a framework to encode verify and robustly train frequency-characterised specifications. Our approach defines input specifications in the Fourier domain and propagates them using an inverse Fourier-transform encoding network prepended to the network to be verified. We demonstrate the ability of our framework to encode perturbations across the spectrum from the low-frequency intensity changes up to the high-frequency white noise kernel-based and domain changes. We then use SoA verifiers to verify differently-trained networks for non-trivial robustness guarantees against some of these practically relevant specifications. Finally, we integrate our framework within existing certified training schemes to enhance network's verified robustness against the proposed specifications by up to 50%.

Introducing Pyra: A High-level Linter for Data Science Software. European Conference on Machine Learning

Participants: Greta Dolcetti, Vincenzo Arceri, Antonella Mensi, Enea Zaffanella, Caterina Urban, Agostino Cortesi.

In [21], we present Pyra, a static analysis tool that aims at detecting code smells in data science workflows. Our goal is to capture potential issues, focusing on misleading visualizations, challenges for reproducibility, as well as misleading, unreliable or unexpected results.

Hallucination-Resilient LLM-Driven Sound and Tunable Static Analysis: A Case of Higher-Order Control-Flow Analysis

Participants: Guannan Wei, Zhuo Zhang, Caterina Urban.

In [27], we argue that soundness remains essential for LLM-driven static analysis and discuss hallucination-resilient approaches in combining LLMs with static analysis that ensure soundness while improving precision. We propose to use LLMs as a way of meta-analysis and investigate this approach in higher-order control-flow analysis, building on the abstracting abstract machine framework and delegating abstract address allocation to an LLM. Our analyzer Ilmaam maintains soundness regardless of LLM behavior, while adaptively tuning analysis precision. We report promising preliminary results and outline broader opportunities for sound LLM-driven analysis.

8.10 Concurrency

Categorical Continuation Semantics for Concurrency

Participants: Flavien Breuvert, Hugo Paquet.

Continuation semantics for simple programming languages can be axiomatized as a dialogue category: a symmetric monoidal category equipped with a negation operation. This axiomatization makes clear the relationship between game semantics, CPS transformations, and continuation monads.

In [18], we extend dialogue categories with 2-categorical structure and concurrent primitives. This is inspired by a recent analysis of concurrency based on 2-categorical monads. We show that the fine-grained structure of dialogue categories, not generally available in other semantic models, can be exploited to give a type to concurrent primitives join and fork. Our main theorem is that this simple axiomatization induces a concurrent continuation 2-monad. We also show that this framework is expressive beyond call-by-value monadic programming.

The definitions in this paper are illustrated by concrete constructions in concurrent game semantics, and our results give a formal categorical basis for concurrent strategies. From a more practical perspective, our approach suggests a candidate target language for linear CPS transformations of concurrent programming languages.

From Thin Concurrent Games to Generalized Species of Structures

Participants: Pierre Clairambault, Federico Olimpieri, Hugo Paquet.

Two families of denotational models have emerged from the semantic analysis of linear logic: dynamic models, typically presented as game semantics, and static models, typically based on a category of relations. In this paper we introduce a formal bridge between a dynamic model and a static model: the model of thin concurrent games and strategies, based on event structures, and the model of generalized species of structures, based on distributors. In [12], we focus on the two-dimensional nature of the dynamic-static relationship, which we formalize with double categories and bicategories.

In the first part of the paper, we construct a symmetric monoidal oplax functor from linear concurrent strategies to distributors. We highlight two fundamental differences between the two models: the composition mechanism, and the representation of resource symmetries.

In the second part of the paper, we adapt established methods from game semantics (visible strategies, payoff structure) to enforce a tighter connection between the two models. We obtain a Cartesian closed pseudofunctor, which we exploit to shed new light on recent results in the theory of the λ -calculus.

8.11 Application to Distributed Applications

CESAn: A Core Erlang Semantics Analyser

Participants: Aurélie Kong Win Chang, Jérôme Feret, Gregor Göbller.

In the concurrent distributed language Erlang, processes interact through message passing and signals. One of the main sources of non-determinism that make Erlang programs difficult to debug, is the order in which messages are handled. In [16], we present a prototype tool called CESAn 7.1.3 that implements a small-step semantics that faithfully represents message handling in Core Erlang, enabling the user to investigate the causes of non-determinism in Erlang stemming from message handling. We see CESAn as a building block for debugging and analysis tools for Erlang.

Causal debugging

Participants: Aurélie Kong Win Chang , Jérôme Feret, Gregor Göbller.

During the development of a system, big amounts of time and energy are spent in debugging. In concurrent systems, several programs can run at the same time, and interact together. This increases the risks of bugs, creates new ones directly born from the concurrent nature of the system, and increases the amount of information which has to be parsed while debugging. On its side, Erlang is a language dedicated, since its birth, to be used in the creation of huge concurrent systems.

In her PhD thesis [29], Aurélie Kong Win Chang, investigated the construction of causal explanations in a debugging context, for Erlang systems, based on a new semantics of Erlang which included the handling of signals.

8.12 Application to Computational Systems Biology

On Model Reductions of Boolean Networks

Participants: Claudine Chaouiya, Jérôme Feret, Patricia Tenera Roxo.

Boolean networks are a class of qualitative models used extensively to model regulatory interactions driving biological processes. While structurally simple, their dynamics can be quite complex. One recurrent issue in studying the dynamics of these models is state explosion. To tackle this problem model reduction techniques are employed. These techniques aim at producing smaller models that behave approximately as the original model.

In [20], we introduce a new reduction that ensures that if there is a path between two states in the initial model, there is also a path between their reduced counterparts. We use Abstract Interpretation, a unifying framework to compare semantics of models at different levels of abstraction, to formally relate the behaviors of the initial and reduced models. We analyze both the newly proposed reduction and Naldi et al.'s reduction through the lenses of this framework.

Reachability Analysis for Parametric Rule-Based Models

Participants: Jérôme Feret, Rebecca Ghidini.

Biological system modeling is an iterative process where uncertainties may arise, especially in the early stages of the modeling. Static analysis tools are needed during each stage of the modeling to help modelers detect unexpected behaviors early by automatically inferring properties about the model. However, the rule-based modeling language Kappa and its static analysis tool KaSa 7.1.6 currently lack support for incomplete models.

In [23], we extend Kappa to support incomplete models, where some rules are considered or not depending on the value of some boolean parameters. We also generalize the current reachability analysis of the static analyzer KaSa to these parametric models, establishing relationships between properties and parameter values. Finally, we implement and evaluate our approach on example models.

Model reduction of infinite rule-based models

Participants: Jérôme Feret.

In [22], we propose a systematic approach to approximate the behavior of models of unbounded polymers. Our technique consists in discovering time-dependent ranges for the quantities of some patterns of interest. These ranges are obtained by approximating the state of the system by a hyper-box (i.e. a rectangle in n -dimensions), with differential equations defining the evolution of the coordinates of each of its hyper-faces (e.g. its $(n - 1)$ -faces). Each equation pessimistically bounds the time derivative of the corresponding coordinate, when the state of the system ranges over this hyper-face. To synthesize these bounds, we focus on the models written in the Kappa language. Kappa provides symbolic equalities and inequalities which intentionally may be understood as algebraic constructions over patterns, and extensionally as sound properties about the quantities of the biomolecular species that contain these patterns.

9 Partnerships and cooperations

Participants: Jérôme Feret, Xavier Rival, Caterina Urban.

9.1 International initiatives

9.1.1 Inria associate team not involved in an IIL or an international program

AISAPPL

Title: Abstract Interpretation-based Static Analysis for Probabilistic Programming Languages

Duration: 2023 ->

Coordinator: Hongseok Yang (hongseok00@gmail.com) and Xavier Rival

Partners:

- Korea Advanced Institute of Science and Technology Daejeon (Corée du Sud)

Inria contact: Xavier Rival

Summary: Probabilistic programming languages describe computations over probability distributions and are increasingly used in various modeling applications as well as machine learning applications. However their complex semantics makes correctness of programs difficult to guarantee for developers, and their computation complexity remains high. To address these issues, we propose to develop techniques based on abstract interpretation static analysis, so as to guarantee in a static maner correctness properties and to optimise probabilistic inference

9.2 National initiatives

9.2.1 ANR EMASS

- Title: Analyse Mémoire Efficace de Logiciel Système
- Type: ANR appel 2022
- Defi: CE39 - Sécurité globale, résilience et gestion de crise, cybersécurité
- Instrument: ANR grant
- Duration: 2023 - 2027
- Coordinator: CEA Saclay
- Others partners: INRIA (France), Thalés (France)
- Inria contact: Xavier Rival

- **Abstract:** The goal of this project is to develop and integrate static analysis techniques that target memory properties for low level code (such as operating system code), in order both to establish safety and security properties. As part of this project, **Antique** is carrying out research on the analysis of programs using complex data-structures.

9.2.2 ANR ForML

- **Title:** Formally Certified Reasoning in Machine Learning
- **Type:** ANR appel 2023
- **Defi:** CE25 - Software sciences and engineering - Multi-purpose communication networks, high-performance infrastructures
- **Instrument:** ANR grant
- **Duration:** 2023 - 2027
- **Coordinator:** Université Toulouse 3 - Paul Sabatier
- **Others partners:** Inria (France), Institut National Polytechnique Toulouse (France), Sorbonne Université (France)
- **Inria contact:** Caterina Urban
- **Abstract:** The ongoing advances in machine learning (ML) foretell an ever-increasing range of practical uses of ML models. However, in domains deemed high-risk at the EU level, but also in safety-critical domains, the deployment of complex ML models should be underpinned by reasoning tools that are rigorous and efficient, and which are thus capable of identifying possible limitations of such ML models. Concrete examples include whether the ML model is adequately robust, whether it does not exhibit bias, but also whether its actions can be explained so as to be understood by a human decision maker or automatically validated. The ForML project proposes to make fundamental inroads in advancing the state of the art in rigorous approaches for reasoning about ML models. The ForML project proposes to exploit well-known hallmarks from software analysis, including abstract interpretation and counterexample-guided abstraction refinement, to promote a new generation of tools for rigorous ML reasoning, in explainability, fairness and robustness, which are both automated and efficient. Furthermore, the ForML project will also pioneer the certification of such tools by exploiting proof assistants.

9.2.3 PEPR SecurEval

- **Title:** SecurEval, Improving Digital Systems Security Evaluation
- **Type:** PEPR
- **Defi:** PEPR Cybersécurité
- **Instrument:** PEPR
- **Duration:** 2022 - 2028
- **Coordinator:** CEA Saclay
- **partners:** CNRS, INRIA, CEA, INP Grenoble, Supelec, UGA, Université Paris Saclay, Université Sorbonne nouvelle
- **Inria contact:** Xavier Rival and Jérôme Feret
- **Abstract:** This project targets methods to improve the security of software, using programming languages techniques (static analysis, testing, programming languages). It gathers a large number of academic partners (CNRS, INRIA, CEA, INP Grenoble, Supelec, UGA, Université Paris Saclay, Université Sorbonne nouvelle). As part of this project, **Antique** is investigating static analysis techniques for the verification of safety and security properties.

9.2.4 PEPR SAIF

- Title: PEPR SAIF
- Type: PEPR
- Defi: PEPR IA
- Instrument: PEPR
- Duration: 2022 - 2028
- Coordinator: CEA Saclay
- partners: INRIA Paris (project team **Antique**), INRIA Saclay (project team TAU), and INRIA Rennes (project team SuMo), Université Paris-Saclay (Formal Methods Laboratory, LMF), École Polytechnique (Computer Science Laboratory, LIX), CEA-List (Software Safety & Security Lab, LSL), and Université de Bordeaux (Bordeaux Computer Science Laboratory)
- Inria contact: Caterina Urban
- Abstract: SAIF is a project led by Caterina Urban within the PEPR IA. The consortium includes INRIA Paris (project team **Antique**), INRIA Saclay (project team TAU), and INRIA Rennes (project team SuMo), as well as Université Paris-Saclay (Formal Methods Laboratory, LMF), École Polytechnique (Computer Science Laboratory, LIX), CEA-List (Software Safety & Security Lab, LSL), and Université de Bordeaux (Bordeaux Computer Science Laboratory).
The overall goal of SAIF is to use the vast knowledge accumulated over decades in formal methods to rethink them and address the novel safety concerns raised by machine learning-based systems.

10 Dissemination

10.1 Promoting scientific activities

Participants: Bernadette Charron-Bost, Jérôme Feret, Hugo Paquet, Xavier Rival, Caterina Urban.

10.1.1 Scientific events: organisation

General chair, scientific chair

- Caterina Urban served as the general chair of the 20th Conference on Integrated Formal Methods (iFM 2025).
- Hugo Paquet is serving as the scientific chair of the Languages for Inference workshop (LAFI 2026).

Member of the steering committee

- Caterina Urban is a Member of the Steering Committee of Static Analysis Symposium (SAS).
- Caterina Urban is a Member of the Steering Committee of State Of the Art in Program Analysis (SOAP).
- Caterina Urban is a member of the Steering Committee of the series of Summer Schools on Foundations of Programming and Software Systems (FoPSS).
- Caterina Urban is a member of the ETAPS Executive Board.

10.1.2 Scientific events: selection

Member of the conference program committees

- Jérôme Feret served a Member of the Program Committee of PADS 2025 (ACM SIGSIM International Conference on Principles of Advanced Discrete Simulation).
- Jérôme Feret served a Member of the Program Committee of CMSB 2025 (conference on Computational Methods in Systems Biology).
- Jérôme Feret served a Member of the Program Committee of iFM 2025 (International Conference on integrated Formal Methods).
- Jérôme Feret is serving as a Member of the Program Committee of PADS 2026 (ACM SIGSIM International Conference on Principles of Advanced Discrete Simulation).
- Jérôme Feret is serving as a Member of the Program Committee of CMSB 2026 (conference on Computational Methods in Systems Biology).
- Hugo Paquet served a Member of the Program Committee of JFLA 2026 (Journées Françaises des Langages Applicatifs).
- Hugo Paquet is serving as a Member of the Program Committee of LICS 2026 (ACM/IEEE Symposium on Logic in Computer Science).
- Hugo Paquet served a Member of the Program Committee of LAFI 2026 (Languages for Inference Workshop).
- Hugo Paquet served a Member of the Program Committee of GALOP 2025 (International Workshop on Games and Logic for Programming).
- Xavier Rival served a Member of the Program Committee of POPL 2025 (ACM SIGPLAN Symposium on Principles of Programming Languages).
- Xavier Rival served a Member of the Program Committee of VMCAI 2025 (Conference on Verification, Model Checking and Abstract Interpretation).
- Xavier Rival served a Member of the Program Committee of CSF 2025 (Conference on Security Foundations).
- Xavier Rival served a Member of the Program Committee of JFLA 2025 (Journées Françaises des Langages Applicatifs).
- Xavier Rival served a Member of the Program Committee of SAS 2025 (Static Analysis Symposium).
- Xavier Rival is serving as a Member of the Program Committee of CSF 2026 (Conference on Security Foundations).
- Xavier Rival is serving as a Member of the Program Committee of ICFP 2026 (ACM SIGPLAN International Conference on Functional Programming).
- Caterina Urban served a Member of the Program Committee of POPL 2025 (ACM SIGPLAN Symposium on Principles of Programming Languages).
- Caterina Urban served a Member of the Program Committee of FoSSaCS 2025 (International Conference on Foundations of Software Science and Computation Structures).
- Caterina Urban served a Member of the Program Committee of SAS 2025 (Static Analysis Symposium).
- Caterina Urban is serving as a Member of the Program Committee of CAV 2026 (International Conference on Computer-Aided Verification).

- Caterina Urban is serving as a Member of the Program Committee of PLDI 2026 (ACM SIGPLAN Conference on Programming Language Design and Implementation).
- Caterina Urban is serving as a Member of the Program Committee of OOPSLA 2026 (Object-oriented Programming, Systems, Languages, and Applications).

Reviewer

- Jérôme Feret served as a reviewer for LICS 2025 (ACM/IEEE Symposium on Logic in Computer Science)
- Hugo Paquet served as a reviewer for FoSSaCS 2026 (International Conference on Foundations of Software Science and Computation Structures)

10.1.3 Journal

Member of the editorial boards

- Jérôme Feret is serving as Associated Editor for **TOMACS** (Transactions on Modeling and Computer Simulation).
- Caterina Urban is serving as Associated Editor for **TOPLAS** (Transactions on Programming Languages and Systems).

Reviewer - reviewing activities

- Jérôme Feret served as a reviewer for **LMCS** (Logical Methods in Computer Science)
- Hugo Paquet served as a reviewer for **MSCS** (Mathematical Structures in Computer Science)
- Xavier Rival served as a reviewer for **ACM Transactions on Probabilistic Machine Learning** (TOPML)
- Xavier Rival served as a reviewer for **Journal of Formal Aspects of Computing** (FAC)
- Xavier Rival served as a reviewer for **Journal of Applied Logics** (JAL)
- Xavier Rival served as a reviewer for **Transactions on Privacy and Security** (TOPS)

10.1.4 Invited talks

- Xavier Rival gave an invited talk at SOAP (State Of the Art Program Analysis, satellite event of PLDI 2026), in July 2026 (Seoul).
- Xavier Rival gave an invited talk at SAS (Static Analysis Symposium, SPLASH 2026) in October 2026 (Singapore).

10.1.5 Leadership within the scientific community

- Xavier Rival is a member of the IFIP Working Group 2.4 on Software Implementation Technologies.

10.1.6 Scientific expertise

- Bernadette Charron-Bost is a member of the administration board of the Laboratoire d'Excellence CIMI de l'Université de Toulouse.
- Bernadette Charron-Bost is a member of the steering committee of the Blockchain X-CapGemini chair.
- Jérôme Feret is a member of the Scientific and Pedagogical Advisory Board of the Interdisciplinary Center for Strategic Studies (CIENS) de l'École normale supérieure.
- Caterina Urban is a member of the Scientific Advisory Board of the Laboratoire Méthodes Formelles (LMF) de l'Université Paris-Saclay.

10.1.7 Recruiting Juries

- Jérôme Feret served in the hiring committee for an Associate Professor in Computer Science at Université de Marseille in 2025.
- Caterina Urban served in the hiring committee for an Associate Professor in Computer Science at Université de Lille in 2025.
- Caterina Urban served in the "admissibility" jury for INRIA researcher positions (ISFP/CRCN) for the center "de l'Université de Lorraine" in 2025.

10.1.8 Research administration

- Bernadette Charron-Bost is the gender-equality referent of the DIENS for the CNRS (COREGAL network).
- Jérôme Feret is a member of the PhD Review Committee (CSD) of Inria Paris.
- Jérôme Feret is dean of study of the department of computer science of École normale supérieure.
- Jérôme Feret is member of the laboratory board of the department of computer sciences of École normale supérieure.
- Jérôme Feret is participating to the INRIA mentoring program as a mentor.
- Xavier Rival is the head of the department of computer sciences of École normale supérieure and of the UMR (Unité Mixte de Recherche) 8548 (DIENS).
- Xavier Rival is a member of the laboratory board of the department of computer sciences of École normale supérieure.
- Xavier Rival is a member of the Bureau du Comité des Projets of the INRIA Paris Research Center.

10.2 Teaching - Supervision - Juries - Educational and pedagogical outreach

10.2.1 Teaching

- Licence:
 - Jérôme Feret and Xavier Rival (lectures), and Charles de Haro (tutorials), "Semantics and Application to Verification", 36h, L3, at École Normale Supérieure, France.
- Master:
 - Bernadette Charron-Bost, "Fundamentals in distributed computing", 60h, M1 Ecole Polytechnique Master.
 - Bernadette Charron-Bost, "Consensus problems", 30h, M2 Ecole Polytechnique Master.
 - Jérôme Feret, Antoine Miné, Xavier Rival, and Caterina Urban, "Abstract Interpretation: application to verification and static analysis", 72h, M2. Parisian Master of Research in Computer Science (MPRI), France.
 - Jérôme Feret and François Fages, "Biochemical Programming", 24h, M2. Parisian Master of Research in Computer Science (MPRI), France.
 - Hugo Paquet (tutorials), "Lambda-calculus and categories", 18h, M1, at École Normale Supérieure, France.
 - Hugo Paquet, "Probabilistic programming languages", 3h, M2, Parisian Master of Research in Computer Science (MPRI), France.
 - Xavier Rival, "Probabilistic programming languages", 3h, M2, Parisian Master of Research in Computer Science (MPRI), France.

10.2.2 Entrance competition juries

- Bernadette Charron-Bost is the co-head of the university track of the École normale supérieure entrance competition for the Computer Science Department.
- Hugo Paquet was a jury member for the Computer Science oral session of the ENS entrance competition (Ulm–Saclay–Lyon–Rennes).

10.2.3 Supervision

- PhD in progress: Jérôme Boillot, Static Analysis of the setting of expanded memory in a dedicated operating system, started in 2022 and supervised by Jérôme Feret.
- PhD in progress: Patricia Roxo, Impact of qualitative variations on the dynamics of Boolean networks, started in 2024 and supervised by Claudine Chaouya (I2M - Marseille) and Jérôme Feret.
- PhD in progress: Rebecca Ghidini, Incremental analysis of parametric biological models, started in 2025 and supervised by Jérôme Feret.
- PhD in progress: Serge Lechenne, Petri net foundations of probabilistic programming, started in 2025 and supervised by Hugo Paquet.
- PhD in progress: Ariadne Si Suo, Probabilistic Programming Semantics, started in 2025 and supervised by Hugo Paquet and Christine Tasson and Thomas Ehrhard.
- PhD in progress: Valentin Barbazo, Static analysis of parallel programs manipulating complex data structures, started in 2023 and supervised by Xavier Rival.
- PhD in progress: Charles De Haro, Abstract interpretation-based static analysis of synchronous data-flow languages, started in 2024 and supervised by Marc Pouzet (INRIA Paris - Project team Parkas) and Xavier Rival.
- PhD in progress: Yoann Bouniard, Analyse statique pour composants de systèmes d'exploitation, started in 2025 and supervised by Xavier Rival.
- PhD in progress: Jérôme Fauchaux, Combinaison de sous et surapproximations de la memoire pour l'analyse de code bas-niveau, started in 2025 and supervised by Matthieu Lemerre (CEA) and Xavier Rival.
- PhD in progress: Naïm Moussaoui-Remil, Abstract Interpretation-Based Static Analyses for Liveness and Security Properties of Programs, started in 2023 and supervised by Caterina Urban.
- PhD in progress: Priyanka Maity, Abstract Interpretation for Explainable Artificial Intelligence, started in 2025 and supervised by Caterina Urban.
- PhD defended : Serge Durand, Over-Approximating Neural Networks for Verification, Robustness, and Explainability, started in 2021 and supervised by Zakaria Chihani (CEA/List) and Caterina Urban (defended the 19th December 2025).
- PhD defended : Aurélie Kong Win Chang, Abstractions for causal analysis and explanations in concurrent programs, started in 2021 and supervised by Gregor Gössler (INRIA Grenoble - Rhône Alpes, Project team Spades) and Jérôme Feret (defended the 16th May 2025).

10.2.4 PhD Juries

- Jérôme Feret served in the committee of the PhD Defense Charles Babu Mamidesetti as a reviewer, CEA-List (December 2025)
- Caterina Urban served in the committee of the PhD Defense Charles Babu Mamidesetti as an examiner, CEA-List (December 2025)

- Caterina Urban served in the committee of the PhD Defense Anna Becchi as a reviewer, Università di Trento, Italy (September 2025)
- Caterina Urban served in the committee of the PhD Defense John Törnblom as a reviewer, Linköping University, Sweden (August 2025)
- Caterina Urban served in the committee of the PhD Defense Linpeng Zhang as a reviewer, University College London, UK (July 2025)
- Caterina Urban served as a reviewer of the PhD manuscript of Kumar Kalita without participating to the PhD Defense committee, the Indian Institute of Technology Kanpur, India (2025)
- Xavier Rival served in the committee of the HdR Defense Caterina Urban as an examiner, ENS Paris/PSL (September 2025)
- Xavier Rival served in the committee of the PhD Defense Antoine Houssais as an examiner, ENS Paris/PSL (November 2025)
- Xavier Rival served in and chaired the committee of the HdR Defense Matthieu Lemerre, Université Paris-Saclay (November 2025)
- Xavier Rival served in and chaired the committee of the PhD Defense Antonin Reitz, ENS Paris/PSL (December 2025)
- Xavier Rival served in and chaired the committee of the PhD Defense Julien Simonnet, Université Paris-Saclay (December 2025)

10.3 Popularization

10.3.1 Specific official responsibilities in science outreach structures

- Jérôme Feret hosted a group of twelve college students for three hours in June. These students has been invited by Inria to visit few research teams in the context of their observation internships.
- Jérôme Feret hosted a group of twelve high school students for three hours in December. These students has been invited by Department of Mathematics of École normale supérieure, to visit few research teams in the context of their observation internships.
- Jérôme Feret hosted a group of nine high school students for three hours in December. These students has been invited by Inria to visit few research teams in the context of their observation internships.
- Xavier Rival hosted a group of twelve high school students for one hour in December. These students has been invited by Department of Mathematics of École normale supérieure, to visit few research teams in the context of their observation internships.

11 Scientific production

11.1 Major publications

- [1] J. Bertrane, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné and X. Rival. ‘Static Analysis and Verification of Aerospace Software by Abstract Interpretation’. In: *Proceedings of the American Institute of Aeronautics and Astronautics (AIAA Infotech@Aerospace 2010)*. Atlanta, Georgia, USA: American Institute of Aeronautics and Astronautics, 2010.
- [2] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival. ‘A Static Analyzer for Large Safety-Critical Software’. In: *PLDI: Conference on Programming Language Design and Implementation*. Association for Computer Machinery (ACM), 2003, pp. 196–207. doi: [10.1145/781131.781153](https://hal.science/hal-00128135). URL: <https://hal.science/hal-00128135>.

- [3] *Element-free probability distributions and random partitions*. LICS '24: 39th Annual ACM/IEEE Symposium on Logic in Computer Science. ACM, 8th July 2024, pp. 1–14. doi: [10.1145/3661814.3662131](https://doi.org/10.1145/3661814.3662131). URL: <https://hal.science/hal-04892070>.
- [4] M. Champion, M. Dalla Preda, R. Giacobazzi and C. Urban. ‘Monotonicity and the Precision of Program Analysis’. In: *Proceedings of the ACM on Programming Languages* 8.POPL (5th Jan. 2024), pp. 1629–1662. doi: [10.1145/3632897](https://doi.org/10.1145/3632897). URL: <https://inria.hal.science/hal-04423578>.
- [5] B. Charron-Bost and A. Schiper. ‘The Heard-Of model: computing in distributed systems with benign faults’. In: *Distributed Computing* 22.1 (8th July 2009), pp. 49–71. doi: [10.1007/S00446-009-0084-6](https://doi.org/10.1007/S00446-009-0084-6). URL: <https://hal.science/hal-04891086> (cit. on p. 9).
- [6] P. Cousot. ‘Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation’. In: *Theoretical Computer Science* 277.1–2 (2002), pp. 47–103.
- [7] J. Feret, V. Danos, J. Krivine, R. Harmer and W. Fontana. ‘Internal coarse-graining of molecular systems’. In: *Proceedings of the National Academy of Sciences of the United States of America* 106.16 (3rd Apr. 2009). doi: [10.1073/pnas.0809908106](https://doi.org/10.1073/pnas.0809908106). URL: <https://inria.hal.science/inria-00528330>.
- [8] L. Mauborgne and X. Rival. ‘Trace Partitioning in Abstract Interpretation Based Static Analyzers’. In: *Proceedings of the 14th European Symposium on Programming (ESOP'05)*. Ed. by M. Sagiv. Vol. 3444. Lecture Notes in Computer Science. Springer-Verlag, 2005, pp. 5–20.
- [9] X. Rival. ‘Symbolic Transfer Functions-based Approaches to Certified Compilation’. In: *Conference Record of the 31st Annual ACM SIGPLAN—SIGACT Symposium on Principles of Programming Languages*. ACM Press, New York, United States, 2004, pp. 1–13.
- [10] C. Urban, M. Christakis, V. Wüstholtz and F. Zhang. ‘Perfectly Parallel Fairness Certification of Neural Networks’. In: *Proceedings of the ACM on Programming Languages* 4.OOPSLA (13th Nov. 2020), pp. 1–30. doi: [10.1145/3428253](https://doi.org/10.1145/3428253). URL: <https://inria.hal.science/hal-03091870>.

11.2 Publications of the year

International journals

- [11] M. Champion, M. Dalla Preda, R. Giacobazzi and C. Urban. ‘A Logic for the Imprecision of Abstract Interpretations’. In: *Proceedings of the ACM on Programming Languages* POPL 2026 (2025). URL: <https://hal.science/hal-05402142>. In press (cit. on p. 20).
- [12] P. Clairambault, F. Olimpieri and H. Paquet. ‘From Thin Concurrent Games to Generalized Species of Structures (Extended Version)’. In: *Logical Methods in Computer Science* Volume21, Issue4 (28th Oct. 2025). doi: [10.46298/lmcs-21\(4:12\)2025](https://doi.org/10.46298/lmcs-21(4:12)2025). URL: <https://hal.science/hal-05383180> (cit. on p. 23).
- [13] A. De Palma, S. Durand, Z. Chihani and C. Urban. ‘On Using Certified Training towards Empirical Robustness’. In: *Transactions on Machine Learning Research Journal* (2025). URL: <https://inria.hal.science/hal-05042448> (cit. on p. 22).
- [14] S. Lim, H. Lim, W. Lee, X. Rival and H. Yang. ‘Optimising Density Computations in Probabilistic Programs via Automatic Loop Vectorisation’. In: *Proceedings of the ACM on Programming Languages* 10.POPL (8th Jan. 2026), pp. 597–627. doi: [10.1145/3776663](https://doi.org/10.1145/3776663). URL: <https://hal.science/hal-05468306> (cit. on p. 20).
- [15] C. Urban, P. Subotić and F. Drobnjaković. ‘Static Analysis by Abstract Interpretation Against Data Leakage in Machine Learning’. In: *Science of Computer Programming* 246 (Dec. 2025), p. 103338. doi: [10.1016/j.scico.2025.103338](https://doi.org/10.1016/j.scico.2025.103338). URL: <https://inria.hal.science/hal-05108093> (cit. on p. 21).

Invited conferences

- [16] A. Kong Win Chang, J. Feret and G. Gössler. ‘CESAn: A Core Erlang Semantics Analyser’. In: *Components Operationally: Reversibility and System Engineering. Essays Dedicated to Jean-Bernard Stefani on the Occasion of His 65th Birthday*. DisCoTec 2025 - 20th International Federated Conference on Distributed Computing Techniques / CORSE - Components Operationally: Reversibility and System Engineering. Vol. LNCS-16065. Lecture Notes in Computer Science. Lille, France: Springer Nature Switzerland, 18th Oct. 2025, pp. 103–115. doi: [10.1007/978-3-031-99717-4_6](https://doi.org/10.1007/978-3-031-99717-4_6). URL: <https://inria.hal.science/hal-05353119> (cit. on pp. 9, 23).

International peer-reviewed conferences

- [17] J. Boillot and J. Feret. ‘Abstraction of memory block manipulations by symbolic loop folding’. In: *European Symposium on Programming 2025*. ESOP 2025 - 34th European Symposium on Programming. Vol. 15694. Hamilton, Canada: Springer, Cham, 2025, pp. 117–143. doi: [10.1007/978-3-031-91118-7_5](https://doi.org/10.1007/978-3-031-91118-7_5). URL: <https://inria.hal.science/hal-04853849> (cit. on p. 17).
- [18] F. Breuvar and H. Paquet. ‘Categorical Continuation Semantics for Concurrency’. In: *FSCD 2025 - 10th International Conference on Formal Structures for Computation and Deduction*. Vol. 10th International Conference on Formal Structures for Computation and Deduction (FSCD 2025). Birmingham (UK), United Kingdom: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi: [10.4230/LIPIcs.FSCD.2025.10](https://doi.org/10.4230/LIPIcs.FSCD.2025.10). URL: <https://inria.hal.science/hal-05476463> (cit. on p. 23).
- [19] M. Champion, I. Mastroeni and C. Urban. ‘Relating Distances and Abstractions: An Abstract Interpretation Perspective’. In: *SAS 2025 - 32nd Static Analysis Symposium*. Singapore, Singapore, 13th Oct. 2025. URL: <https://inria.hal.science/hal-05207631> (cit. on pp. 13, 18).
- [20] C. Chaouiya, J. Feret and P. Roxo. ‘On Model Reductions of Boolean Networks’. In: *23rd International Conference on Computational Methods in Systems Biology, CMSB 2025*. Vol. 15959. Computational Methods in Systems Biology. Lyon, France: Springer Nature, Aug. 2025, p. 19. doi: [10.1007/978-3-032-01436-8_3](https://doi.org/10.1007/978-3-032-01436-8_3). URL: <https://inria.hal.science/hal-05115042> (cit. on p. 24).
- [21] G. Dolcetti, V. Arceri, A. Mensi, E. Zaffanella, C. Urban and A. Cortesi. ‘Introducing Pyra: A High-level Linter for Data Science Software’. In: *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases - ECML PKDD 2025*. Porto, Portugal, 15th Sept. 2025. URL: <https://inria.hal.science/hal-05207628> (cit. on p. 22).
- [22] J. Feret. ‘Model reduction of infinite rule-based models’. In: *PADS 25 - 39th ACM SIGPLAN Principle of Advanced Discrete Simulation*. Santa Fe (NM), United States: ACM, 2025. URL: <https://inria.hal.science/hal-05002272> (cit. on p. 25).
- [23] J. Feret and R. Ghidini. ‘Reachability Analysis for Parametric Rule-Based Models’. In: *Computational Methods in Systems Biology*. 23rd International Conference on Computational Methods in Systems Biology, CMSB 2025. Vol. 15959. Computational Methods in Systems Biology. Lyon, France: Springer Nature, 10th Sept. 2025, p. 21. doi: [10.1007/978-3-032-01436-8_9](https://doi.org/10.1007/978-3-032-01436-8_9). URL: <https://inria.hal.science/hal-05121555> (cit. on p. 24).
- [24] H. Hanspal, A. De Palma and A. Lomuscio. ‘Robustness to Perturbations in the Frequency Domain: Neural Network Verification and Certified Training’. In: *Proceedings of the WACV 2025 Workshop on Out-of-Label Hazards in Autonomous Driving*. WACV 2025 - Workshop on Out-of-Label Hazards in Autonomous Driving. Tucson, Arizona, United States, 4th May 2025. URL: <https://hal.science/hal-05330224> (cit. on p. 22).
- [25] W. Lee, M. Lemerre, X. Rival and H. Yang. ‘On the Structure of Abstract Interpreters’. In: *OLIVIERFEST 2025 - Workshop Dedicated to Olivier Danvy on the Occasion of His 64th Birthday*. OLIVIERFEST ’25: Workshop Dedicated to Olivier Danvy on the Occasion of His 64th Birthday. Singapore Singapore, Singapore: ACM, 12th Oct. 2025, pp. 65–71. doi: [10.1145/3759427.3760368](https://doi.org/10.1145/3759427.3760368). URL: <https://hal.science/hal-05468431> (cit. on p. 19).

- [26] N. Moussaoui Remil and C. Urban. ‘Termination Resilience Static Analysis’. In: VMCAI 2026 - 27th International Conference on Verification, Model Checking, and Abstract Interpretation. Rennes, France, 12th Jan. 2026. URL: <https://inria.hal.science/hal-05398150> (cit. on p. 18).
- [27] G. Wei, Z. Zhang and C. Urban. ‘Hallucination-Resilient LLM-Driven Sound and Tunable Static Analysis: A Case of Higher-Order Control-Flow Analysis’. In: LMPL 2025 - 1st International Workshop on Language Models and Programming Languages. Singapore, Singapore, 15th Oct. 2025. DOI: [10.1145/3759425.3763378](https://doi.org/10.1145/3759425.3763378). URL: <https://inria.hal.science/hal-05296479> (cit. on p. 22).

Doctoral dissertations and habilitation theses

- [28] S. Durand. ‘Over-Approximating Neural Networks for Verification, Robustness, and Explainability’. Université Paris-Saclay, 19th Dec. 2025. URL: <https://theses.hal.science/tel-05468098> (cit. on p. 21).
- [29] A. Kong Win Chang. ‘Causal explanations for concurrent programs in Erlang’. Université Grenoble Alpes [2020-....], 16th May 2025. URL: <https://theses.hal.science/tel-05262517> (cit. on pp. 9, 24).
- [30] C. Urban. ‘Static Analyses for the Properties, Programs, and People of Tomorrow’. École Normale Supérieure; Université Paris Sciences et Lettres (PSL), 30th Sept. 2025. URL: <https://theses.hal.science/tel-05467963> (cit. on p. 19).

Reports & preprints

- [31] M. Champion, I. Mastroeni, M. Pasqua and C. Urban. *Abstract Lipschitz Continuity: Combining Semantic and Quantitative Approximations*. 23rd Jan. 2026. URL: <https://inria.hal.science/hal-04935306> (cit. on p. 18).

11.3 Cited publications

- [32] P. Cousot. ‘Constructive design of a hierarchy of semantics of a transition system by abstract interpretation’. In: *Electr. Notes Theor. Comput. Sci.* 6 (1997), pp. 77–102. DOI: [10.1016/S1571-0661\(05\)80168-9](https://doi.org/10.1016/S1571-0661(05)80168-9). URL: [http://dx.doi.org/10.1016/S1571-0661\(05\)80168-9](http://dx.doi.org/10.1016/S1571-0661(05)80168-9) (cit. on p. 9).
- [33] P. Cousot and R. Cousot. ‘Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints’. In: *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM Press, New York, United States, 1977, pp. 238–252 (cit. on p. 8).
- [34] J. Cury, P. T. Roxo, V. Manquinho, C. Chaouiya and P. Monteiro. ‘Computation of Immediate Neighbours of Monotone Boolean Functions’. In: *23rd International Conference on Computational Methods in Systems Biology, CMSB 2025*. Lyon, France: arXiv, Sept. 2025. DOI: [10.48550/arXiv.2407.01337](https://doi.org/10.48550/arXiv.2407.01337). URL: <https://hal.science/hal-05127796> (cit. on p. 13).
- [35] F. Drobnjaković, P. Subotic and C. Urban. ‘An Abstract Interpretation-Based Data Leakage Static Analysis’. In: *18th International Symposium on Theoretical Aspects of Software Engineering*. Guiyang, China, July 2024. URL: <https://inria.hal.science/hal-04556578> (cit. on p. 21).
- [36] A. Kong Win Chang, J. Feret and G. Gössler. ‘A Semantics of Core Erlang with Handling of Signals’. In: *ACM Digital Library*. Seattle WA, United States: ACM, Sept. 2023, pp. 31–38. DOI: [10.1145/3609022.3609417](https://doi.org/10.1145/3609022.3609417). URL: <https://hal.science/hal-04222884> (cit. on p. 9).
- [37] C. Urban. ‘Static Analysis of Data Science Software’. In: *SAS 2019 - 26th Static Analysis Symposium*. Ed. by B.-Y. E. Chang. Porto, Portugal: Springer, Oct. 2019, pp. 17–23. DOI: [10.1007/978-3-030-32304-2_2](https://doi.org/10.1007/978-3-030-32304-2_2). URL: <https://hal.inria.fr/hal-02397699> (cit. on p. 13).