RESEARCH CENTRE

Inria Paris Center

IN PARTNERSHIP WITH:

CNRS, Ecole normale supérieure de Paris

2022

ACTIVITY REPORT

Project-Team

ANTIQUE

Static Analysis by Abstract Interpretation

IN COLLABORATION WITH: Département d'Informatique de l'Ecole Normale Supérieure

DOMAIN

Algorithmics, Programming, Software and Architecture

THEME

Proofs and Verification

Inria

Contents

Project-Team ANTIQUE

Creation of the Project-Team: 2015 April 01

Keywords

Computer sciences and digital sciences

A2. – Software

A2.1. – Programming Languages

A2.1.1. – Semantics of programming languages

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

A2.4. – Formal method for verification, reliability, certification

A2.4.1. – Analysis

A2.4.2. – Model-checking

A2.4.3. – Proofs

A2.6.1. – Operating systems

A4.4. – Security of equipment and software

A4.5. – Formal methods for security

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

1 Team members, visitors, external collaborators

Research Scientists

- Xavier Rival [Team leader, INRIA, Senior Researcher, HDR]
- Bernadette Charron-Bost [LIX, Senior Researcher, HDR]
- Vincent Danos [CNRS, Senior Researcher, HDR]
- Jerome Feret [INRIA, Researcher]
- Caterina Urban [INRIA, Researcher]

PhD Students

- Jérôme Boillot [ENS PARIS, from Sep 2022]
- Serge Durand [CEA]
- Josselin Giet [ENS PARIS]
- Patricio Inzaghi [INRIA]
- Denis Mazzucato [INRIA]
- Louis Penet De Monterno [ECOLE POLY PALAISEAU]
- Albin Salazar [INRIA]
- Ignacio Tiraboschi [INRIA]

Technical Staff

- Thierry Martinez [INRIA]

Interns and Apprentices

- Abhinandan Pal [IIIT Kalyani, India, from Nov 2022]
- Felix Ridoux [INRIA, from Sep 2022]

Administrative Assistants

- Nathalie Gaudechoux [INRIA]
- Meriem Guemair [INRIA]

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.

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 life. 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 recently 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.

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.

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.

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 so as to get a better insight in the underlying mechanisms of these systems.

3 Research program

3.1 Semantics

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.

In addition to the properties that are required to express the proof of the property of interest, it may also be desirable that semantics describe program behaviors in a finer manner, so as to make static analyses easier to design. For instance, it is well known that, when a state property (such as the absence of runtime error) is valid, it can be established using only a state invariant (i.e., an invariant that ignores the order in which states are visited during program executions). Yet searching for trace invariants (i.e., that take into account some properties of program execution history) may make the static

analysis significantly easier, as it will allow it to make finer case splits, directed by the history of program executions. To allow for such powerful static analyses, we often resort to a *non standard semantics*, which incorporates properties that would normally be left out of the concrete semantics.

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, so as 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* [26]. The case of loops requires a bit more work as the concrete semantics typically relies on a fixpoint that may not be computable in finitely many iterations. To achieve a terminating analysis we then use *widening operators* [26], 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 family 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 [25], 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.

Complex systems and software have often times intricate behaviors, leading to executions that are hard to understand for programmers and also difficult to reason about with static analyzers. Shared memory and distributed systems are notorious for being hard to reason about due to the interleaving of actions performed by different processes and the non-determinism of the network that might lose, corrupt, or duplicate messages. Reduction theorems, e.g., Lipton's theorem, have been proposed to facilitate reasoning about concurrency, typically transforming a system into one with a coarse-grained semantics that usually increases the atomic sections. We investigate reduction theorems for distributed systems and ways to compute the coarse-grained counter part of a system automatically. Compared with shared memory concurrency, automated methods to reason about distributed systems have been less investigated in the literature. We take a programming language approach based on high-level programming abstractions. We focus on partially-synchronous communication closed round-based models, introduced in the distributed algorithms community for its simpler proof arguments. The high-level language is compiled into a low-level (asynchronous) programming language. Conversely, systems defined under asynchronous programming paradigms are decompiled into the high-level programming abstractions. The correctness of the compilation/decompilation process is based on reduction theorems (in the spirit of Lipton and Elrad-Francez) that preserve safety and liveness properties.

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 explaining 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 offer 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 very important application domain for our group. First, 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. Second, 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 represent 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 resort 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 so as 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 separate 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, in order to support the analysis of more complex programming language features such as 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 systems 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 Consensus

Fault-tolerant distributed systems provide a dependable service on top of unreliable computers and networks. Famous examples are geo-replicated data-bases, distributed file systems, or blockchains. Fault-tolerant protocols replicate the system and ensure that all (unreliable) replicas are perceived from the outside as one single reliable machine. To give the illusion of a single reliable machine “consensus” protocols force replicas to agree on the “current state” before making this state visible to an outside observer. We are interested in (semi-)automatically proving the total correctness of consensus algorithms in the benign case (messages are lost or processes crash) or the Byzantine case (processes may lie about their current state). In order to do this, we first define new reduction theorems to simplify the behaviors of the system and, second, we introduce new static analysis methods to prove the total correctness of adequately simplified systems. We focus on static analysis based Satisfiability Modulo Theories (SMT) solvers which offers a good compromise between automation and expressiveness. Among our benchmarks are Paxos, PBFT (Practical Byzantine Fault-Tolerance), and blockchain algorithms (Red-Belly, Tendermint, Algorand). These are highly challenging benchmarks, with a lot of non-determinism coming from the interleaving semantics and from the adversarial environment in which correct processes execute, environment that can drop messages, corrupt them, etc. Moreover, these systems were originally designed for a few servers but today are deployed on networks with thousands of nodes. The “optimizations” for scalability can no longer be overlooked and must be considered as integral part of the algorithms, potentially leading to specifications weaker than the so much desired consensus.

4.5 Smart contracts

Blockchain applications in finance have emerged in 2020 as the lead applications. The new field called *decentralised finance* (or also open finance) re-creates basic financial functionalities such as ireeversible and reverible swaps of assets. There are broad goals to our research in this emerging area: structuring contract languages which guarantee good execution properties by construction, and finding mechanisms that foster liquidity.

We are investigating several other problems in decentralised finance: protocols for capital-efficient decentralised exchanges; general convex problems for the optimal routing and arbitrage in the network of exchange platforms; and the economics of the competition between two-sided exchange platforms.

4.6 Static analysis of data science 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 [29].

5 Social and environmental responsibility

5.1 Impact of research results

We are advising several companies such as Bender operating on the Tezos blockchain, think tanks such as the CDC Labchain (Caisse des Dépots), and other informal development groups such as Jaxnet on decentralised finance protocols and mechanism design for consensus incentives.

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

We point out several issues and institutional dysfunctions which impaired and slowed down our team activity, and also negatively affected the general atmosphere in the institute, causing increasing anxiety and strain in the scientific, technical and administrative staff.

- The deployment of the Eksae information system was highly problematic, substantially hindering the activity of our administrative staff, forcing them to tedious duplications and rendering some of their tasks almost impossible. This also had repercussions on researchers, depriving them of a

long-term vision of their budget and generating more constraints and causing delays in purchases and mission reimbursements.

- The presentation of the institute given in our General Direction (DG) addresses and publications, such as the "Rapport d'activité 2021" (Question d'avenir), offered a distorted and unbalanced image of our institute, hiding the primary role of research and emphasising only the most "trendy" and "applicable" research activities, while dismissing many others for which our institute gained its prestigious reputation. The team is deeply concerned to observe the disconnection between this document and the research works carried out in the Institute.
- A deep distress within the staff, as well as an increasing distrust towards the DG, has arisen by effect of the dismissive attitude of the DG towards the consultative bodies of the institute, and particularly towards the Evaluation Commission (CE), whose role is essential for the quality of our hiring and promotion processes as well as for our prospective scientific reflexions, but also towards the Comite des Equipes Projets (CEP), whose formal requests seem to have been ignored by the DG.

As of early 2023, the team is concerned these issues are still not properly addressed.

7 New software and platforms

7.1 New software

7.1.1 APRON

Scientific Description: The APRON library is intended to be a common interface to various underlying libraries/abstract domains and to provide additional services that can be implemented independently from the underlying library/abstract domain, as shown by the poster on the right (presented at the SAS 2007 conference. You may also look at:

Functional Description: The Apron library is dedicated to the static analysis of the numerical variables of a program by abstract interpretation. Its goal is threefold: provide ready-to-use numerical abstractions under a common API for analysis implementers, encourage the research in numerical abstract domains by providing a platform for integration and comparison of domains, and provide a teaching and demonstration tool to disseminate knowledge on abstract interpretation.

URL: http://apron.cri.ensmp.fr/library/

Contact: Antoine Miné

Participants: Antoine Miné, Bertrand Jeannet

7.1.2 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.

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.

Functional 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.

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 Miné, Jerome Feret, Laurent Mauborgne, Patrick Cousot, Radhia Cousot, Xavier Rival

Partners: CNRS, ENS Paris, AbsInt Angewandte Informatik GmbH

7.1.3 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.astreea.ens.fr/

Contact: Patrick Cousot

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

Partners: CNRS, ENS Paris, AbsInt Angewandte Informatik GmbH

7.1.4 ClangML

Keyword: Compilation

Functional Description: ClangML is an OCaml binding with the Clang front-end of the LLVM compiler suite. Its goal is to provide an easy to use solution to parse a wide range of C programs, that can be called from static analysis tools implemented in OCaml, which allows to test them on existing programs written in C (or in other idioms derived from C) without having to redesign a front-end from scratch. ClangML features an interface to a large set of internal AST nodes of Clang , with an easy to use API. Currently, ClangML supports all C language AST nodes, as well as a large part of the C nodes related to C++ and Objective-C.

URL: https://github.com/Antique-team/clangml/tree/master/clang

Contact: Xavier Rival

Participants: Devin Mccoughlin, François Berenger, Pippijn Van Steenhoven

7.1.5 FuncTion

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 Miné, Caterina Urban

7.1.6 HOO

Name: Heap Abstraction for Open Objects

Functional Description: JSAna with HOO is a static analyzer for JavaScript programs. The primary component, HOO, which is designed to be reusable by itself, is an abstract domain for a dynamic language heap. A dynamic language heap consists of open, extensible objects linked together by pointers. Uniquely, HOO abstracts these extensible objects, where attribute/field names of objects may be unknown. Additionally, it contains features to keeping precise track of attribute name/value relationships as well as calling unknown functions through desynchronized separation.

As a library, HOO is useful for any dynamic language static analysis. It is designed to allow abstractions for values to be easily swapped out for different abstractions, allowing it to be used for a wide-range of dynamic languages outside of JavaScript.

Contact: Arlen Cox

Participant: Arlen Cox

7.1.7 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, François Berenger, Huisong Li, Xavier Rival

7.1.8 KAPPA

Name: A rule-based language for modeling interaction networks

Keywords: Systems Biology, Modeling, Static analysis, Simulation, Model reduction

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.

Release Contributions: On line UI, Simulation is based on a new data-structure (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).

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

Contact: Jerome Feret

Participants: Jean Krivine, Jerome Feret, Kim-Quyen Ly, Pierre Boutillier, Russ Harmer, Vincent Danos, Walter Fontana

Partners: ENS Lyon, Université Paris-Diderot, HARVARD Medical School

7.1.9 QUICr

Functional Description: QUICr is an OCaml library that implements a parametric abstract domain for sets. It is constructed as a functor that accepts any numeric abstract domain that can be adapted to the interface and produces an abstract domain for sets of numbers combined with numbers. It is relational, flexible, and tunable. It serves as a basis for future exploration of set abstraction.

Contact: Arlen Cox

Participant: Arlen Cox

7.1.10 Zarith

Functional Description: Zarith is a small (10K lines) OCaml library that implements arithmetic and logical operations over arbitrary-precision integers. It is based on the GNU MP library to efficiently implement arithmetic over big integers. Special care has been taken to ensure the efficiency of the library also for small integers: small integers are represented as Caml unboxed integers and use a specific C code path. Moreover, optimized assembly versions of small integer operations are provided for a few common architectures.

Zarith is currently used in the Astrée analyzer to enable the sound analysis of programs featuring 64-bit (or larger) integers. It is also used in the Frama-C analyzer platform developed at CEA LIST and Inria Saclay.

URL: http://forge.ocamlcore.org/projects/zarith

Contact: Antoine Miné

Participants: Antoine Miné, Pascal Cuoq, Xavier Leroy

7.1.11 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

7.2 New platforms

This year, the team has contributed to its existing platforms.

Participants: .

8 New results

8.1 Shape analysis

Lightweight Shape Analysis based on Physical Types.

Participants: Matthieu Lemerre, Olivier Nicole, Xavier Rival *(correspondant)*.

To understand and detect possible errors in programs manipulating memory, researchers have developed static analyses of various precision levels. Pointer analyses are efficient but too imprecise to prove even simple properties; shape analyses based on the abstract interpretation framework can be very precise, but the heap abstraction that constitutes their internal state can grow exponentially, which poses scalability

problems in practice. In this work, we propose a new memory analysis by abstract interpretation that summarizes the heap by means of a type invariant, using *physical types* which express the byte-level layout of values in memory. We complement this summarizing abstraction with an independent, *focusing* abstraction which refines information about memory regions currently in use, allowing *strong updates* without introducing disjunctions; and a *store buffer* abstraction that allows temporary violation of typing invariants, further improving the precision of the analysis in presence of initialization code. We show that this combination of abstractions suffices to verify memory safety and non-trivial structural invariants in the presence of complex constructs such as type casts, pointer arithmetic and dynamic memory allocation.

This work has been accepted for publishation at VMCAI 2022 [14]. It is also part of the PhD thesis of Olivier Nicole [18].

Sequence Predicates-based Shape Content Abstraction

Participants: Josselin Giet, Xavier Rival *(correspondant)*.

Traditional separation logic-based shape analyses utilize inductive summarizing predicates so as to capture general properties of the layout of data-structures, to verify accurate manipulations of, e.g., various forms of lists or trees. However, they also usually abstract away content properties, which restricts the scope of the possible target properties. In this work, we introduced a novel abstract domain to describe sequences of values of unbounded size, and track constraints on their length and on extremal values contained in them. We defined a reduced product of such a sequence abstraction together with an existing shape abstraction so as to infer complex properties of data-structures. We carried out the implementation of the sequence domain, its integration into a static analyzer targeting C code, and we evaluated the precision of the resulting analysis.

8.2 Verification of security properties on a micro-kernel

No Crash, No Exploit: Automated Verification of Embedded Kernels

Participants: Matthieu Lemerre, Olivier Nicole, Xavier Rival *(correspondant)*.

The kernel is the most safety- and security-critical component of many computer systems, as the most severe bugs lead to complete system crash or exploit. It is thus desirable to guarantee that a kernel is free from these bugs using formal methods, but the high cost and expertise required to do so are deterrent to wide applicability. We proposed a method that can verify both *absence of runtime errors* (i.e. crashes) and *absence of privilege escalation* (i.e. exploits) in embedded kernels from their binary executables. The method can verify the kernel runtime independently from the application, at the expense of *only a few lines* of simple annotations. When given a specific application, the method can verify simple kernels without any human intervention. We demonstrated our method on two different use cases: we used our tool to help the development of a new embedded real-time kernel, and we verifed an existing industrial real-time kernel executable with no modification. Results show that our approach is fast, simple to use, and can prevent real errors and security vulnerabilities.

This work was published at RTAS 2021 [28]. It was also described in Olivier Nicole's PhD thesis [18].

8.3 Static Analysis of Probabilistic Programming Languages and Optimization Algorithms

Towards the verification of semantic assumptions required by probabilistic inference algorithms

Participants: Wonyeol Lee, Hangyeol Wu, Xavier Rival *(correspondant)*, Hongseok Yang.

Probabilistic programming is the idea of writing models from statistics and machine learning using program notations and reasoning about these models using generic inference engines. Recently its combination with deep learning has been explored intensely, which led to the development of so called deep probabilistic programming languages, such as Pyro, Edward and ProbTorch. At the core of this development lie inference engines based on stochastic variational inference algorithms. When asked to find information about the posterior distribution of a model written in such a language, these algorithms convert this posterior-inference query into an optimisation problem and solve it approximately by a form of gradient ascent or descent. We analysed one of the most fundamental and versatile variational inference algorithms, called score estimator or REINFORCE, using tools from denotational semantics and program analysis. We formally expressed what this algorithm does on models denoted by programs, and exposed implicit assumptions made by the algorithm on the models. The violation of these assumptions may lead to an undefined optimisation objective or the loss of convergence guarantee of the optimisation process. We then describe rules for proving these assumptions, which can be automated by static program analyses. Some of our rules use nontrivial facts from continuous mathematics, and let us replace requirements about integrals in the assumptions, such as integrability of functions defined in terms of programs' denotations, by conditions involving differentiation or boundedness, which are much easier to prove automatically (and manually). Following our general methodology, we have developed a static program analysis for the Pyro programming language that aims at discharging the assumption about what we call model-guide support match. Our analysis is applied to the eight representative model-guide pairs from the Pyro webpage, which include sophisticated neural network models such as AIR. It found a bug in one of these cases, and revealed a non-standard use of an inference engine in another, and showed that the assumptions are met in the remaining six cases.

Moreover, we have implemented an analysis for differentiability and other classes of smoothness properties. This analysis can be ran on regular Python programs or on Pyro programs, and verify the differentiability properties required for the sound definition of model guide pairs.,

The basis for this method has been published at POPL 2020 [24].

Smoothness Analysis for Probabilistic Programs with Application to Optimised Variational Inference

Participants: Wonyeol Lee, Xavier Rival *(correspondant)*, Hongseok Yang.

We proposed a static analysis for discovering differentiable or more generally smooth parts of a given probabilistic program, and showed how the analysis can be used to improve the pathwise gradient estimator, one of the most popular methods for posterior inference and model learning. Our improvement increases the scope of the estimator from differentiable models to non-differentiable ones without requiring manual intervention of the user; the improved estimator automatically identifies differentiable parts of a given probabilistic program using our static analysis, and applies the pathwise gradient estimator to the identified parts while using a more general but less efficient estimator, called score estimator, for the rest of the program. Our analysis has a surprisingly subtle soundness argument, partly due to the misbehaviours of some target smoothness properties when viewed from the perspective of program analysis designers. For instance, some smoothness properties, such as partial differentiability and partial continuity, are not preserved by function composition, and this makes it difficult to analyse sequential composition soundly without heavily sacrificing precision. We formulated five assumptions on a target smoothness property, prove the soundness of our analysis under those assumptions, and show that our leading examples satisfy these assumptions. We have also shown that by using information from our analysis instantiated for differentiability, our improved gradient estimator satisfies an important differentiability requirement and thus computes the correct estimate on average (i.e., returns an unbiased estimate) under a regularity condition. Our experiments with representative probabilistic programs in the Pyro language show that our static analysis is capable of identifying smooth parts of those programs accurately, and making our improved pathwise gradient estimator exploit all the opportunities for high performance in those programs.

This work has been accepted for publication at POPL 2023 [10].

8.4 Static Analysis of Jupyter Notebooks

Abstract Interpretation-Based Data Leakage Static Analysis

Participants: Filip Drobnjaković, Pavle Subotić, Caterina Urban.

Data leakage is a well-known problem in machine learning. Data leakage occurs when information from outside the training dataset is used to create a model. This phenomenon renders a model excessively optimistic or even useless in the real world since the model tends to leverage greatly on the unfairly acquired information. To date, detection of data leakages occurs post-mortem using run-time methods. However, due to the insidious nature of data leakage, it may not be apparent to a data scientist that a data leakage has occurred in the first place. For this reason, it is advantageous to detect data leakages as early as possible in the development life cycle. In [19], we propose a novel static analysis to detect several instances of data leakages during development time. We define our analysis using the framework of abstract interpretation: we define a concrete semantics that is sound and complete, from which we derive a sound and computable abstract semantics. We implement our static analysis inside the open-source NBLyzer static analysis framework and demonstrate its utility by evaluating its performance and precision on over 2000 Kaggle competition notebooks.

8.5 Static Analysis of Machine Learning Software

Abstract Interpretation-Based Feature Importance for SVMs

Participants: Abhinandan Pal, Francesco Ranzato, Caterina Urban, Marco Zanella.

In [21], we propose a symbolic representation for support vector machines (SVMs) by means of abstract interpretation, a well-known and successful technique for designing and implementing static program analyses. We leverage this abstraction in two ways: (1) to enhance the interpretability of SVMs by deriving a novel feature importance measure, called abstract feature importance (AFI), that does not depend in any way on a given dataset of the accuracy of the SVM and is very fast to compute, and (2) for verifying stability, notably individual fairness, of SVMs and producing concrete counterexamples when the verification fails. We implemented our approach and we empirically demonstrated its effectiveness on SVMs based on linear and nonlinear (polynomial and radial basis function) kernels. Our experimental results show that, independently of the accuracy of the SVM, our AFI measure correlates much more strongly with the stability of the SVM to feature perturbations than feature importance measures widely available in machine learning software such as permutation feature importance. It thus gives better insight into the trustworthiness of SVMs.

Verifying Attention Robustness of Deep Neural Networks against Semantic Perturbations

Participants: Satoshi Munakata, Caterina Urban, Haruki Yokoyama, Koji Yamamoto, Kazuki Munakata.

It is known that deep neural networks (DNNs) classify an input image by paying particular attention to certain specific pixels; a graphical representation of the magnitude of attention to each pixel is called a saliency-map. Saliency-maps are used to check the validity of the classification decision basis, e.g., it is not a valid basis for classification if a DNN pays more attention to the background rather than the subject of an image. Semantic perturbations can significantly change the saliency-map. In [23, 20], we propose the first verification method for attention robustness, i.e., the local robustness of the changes in the saliency-map against combinations of semantic perturbations. Specifically, our method determines the range of the perturbation parameters (e.g., the brightness change) that maintains the difference

between the actual saliency-map change and the expected saliency-map change below a given threshold value. Our method is based on activation region traversals, focusing on the outermost robust boundary for scalability on larger DNNs. Experimental results demonstrate that our method can show the extent to which DNNs can classify with the same basis regardless of semantic perturbations and report on performance and performance factors of activation region traversals.

ReCIPH: Relational Coefficients for Input Partitioning Heuristic

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

With the rapidly advancing improvements to the already successful Deep Learning artifacts, Neural Networks (NN) are poised to permeate a growing number of everyday applications, including ones where safety is paramount and, therefore, formal guarantees are a precious commodity. To this end, Formal Methods, a long-standing, mathematically-inspired field of research saw an effervescent outgrowth targeting NN and advancing almost as rapidly as AI itself. Without a doubt, the most challenging problem facing this new research direction is the scalability to the evergrowing NN models. [22] stems from this need and introduces Relational Coefficients for Input partitioning Heuristic (ReCIPH), accelerating NN analysis. Extensive experimentation is supplied to assert the added value to two different solvers handling several models and properties (coming, in part, from two industrial use-cases).

8.6 Static analysis for security properties

Sound Symbolic Execution via Abstract Interpretation and its Application to Security

Participants: Tamara Rezk, Xavier Rival *(correspondant)*, Ignacio Tiraboschi.

Symbolic execution is a program analysis technique commonly utilized to determine whether programs violate properties and, in case violations are found, to generate inputs that can trigger them. Used in the context of security properties such as noninterference, symbolic execution is precise when looking for counter-example pairs of traces when insecure information flows are found, however it is sound only up to a bound thus it does not allow to prove the correctness of programs with executions beyond the given bound. By contrast, abstract interpretation-based static analysis guarantees soundness but generally lacks the ability to provide counter-example pairs of traces.

In this work, we propose to weave both to obtain the best of two worlds. We demonstrate this with a series of static analyses, including a static analysis called DSym aimed at verifying noninterference. DSym provides both semantically sound results and the ability to derive counter-example pairs of traces up to a bound. It relies on a combination of symbolic execution and abstract domains inspired by the well known notion of reduced product. We formalize DSym and prove its soundness as well as its relative precision up to a bound. We also provide a prototype implementation of DSym and evaluate it on a sample of challenging examples.

This work has been accepted for publication at VMCAI 2023 [15].

8.7 Reductions between synchronous and asynchronous programming abstractions

Testing consensus implementations using communication closure

Participants: Cezara Drăgoi, Constantin Enea, Burcu Kulahcioglu Ozkan, Rupak Majumdar, Filip Niksic.

Large scale production distributed systems are difficult to design and test. Correctness must be ensured when processes run asynchronously, at arbitrary rates relative to each other, and in the presence of failures, e.g., process crashes or message losses. These conditions create a huge space of executions that is difficult to explore in a principled way. Current testing techniques focus on systematic or randomized exploration of all executions of an implementation while treating the implemented algorithms as black boxes. On the other hand, proofs of correctness of many of the underlying algorithms often exploit semantic properties that reduce reasoning about correctness to a subset of behaviors. For example, the communication-closure property, used in many proofs of distributed consensus algorithms, shows that every asynchronous execution of the algorithm is equivalent to a lossy synchronous execution, thus reducing the burden of proof to only that subset. In a lossy synchronous execution, processes execute in lock-step rounds, and messages are either received in the same round or lost forever—such executions form a small subset of all asynchronous ones.

In [27] we formulate the communication-closure hypothesis, which states that bugs in implementations of distributed consensus algorithms will already manifest in lossy synchronous executions and present a testing algorithm based on this hypothesis. We prioritize the search space based on a bound on the number of failures in the execution and the rate at which these failures are recovered. We show that a random testing algorithm based on sampling lossy synchronous executions can empirically find a number of bugs—including previously unknown ones—in production distributed systems such as Zookeeper, Cassandra, and Ratis, and also produce more understandable bug traces.

8.8 Distributed algorithms

Geometric bounds for convergence rates of averaging algorithms

Participants: Bernadette Charron-Bost.

We developed a generic method for bounding the convergence rate of an averaging algorithm running in a multi-agent system with a time-varying network, where the associated stochastic matrices have a time-independent Perron vector. This method provides bounds on convergence rates that unify and refine most of the previously known bounds. They depend on geometric parameters of the dynamic communication graph such as the normalized diameter or the bottleneck measure. As corollaries of these geometric bounds, we show that the convergence rate of the Metropolis algorithm in a system of n agents is less than $1-0.25n2$ with any communication graph that may vary in time, but is permanently connected and bidirectional. We prove a similar upper bound for the EqualNeighbor algorithm under the additional assumptions that the number of neighbors of each agent is constant and that the communication graph is not too irregular. Moreover our bounds offer improved convergence rates for several averaging algorithms and specific families of communication graphs. Finally we extend our methodology to a time-varying Perron vector and show how convergence times may dramatically degrade with even limited variations of Perron vectors.

This work was published in [9].

Computing Outside the Box: Average Consensus over Dynamic Networks

Participants: Bernadette Charron-Bost, Patrick Lambein-Monette.

Networked systems of autonomous agents, and applications thereof, often rely on the control primitive of average consensus, where the agents are to compute the average of private initial values. To provide reliable services that are easy to deploy, average consensus should continue to operate when the network is subject to frequent and unpredictable change, and should mobilize few computational resources, so that deterministic, low powered, and anonymous agents can partake in the network. In this stringent adversarial context, we investigated the implementation of average consensus by distributed algorithms

over networks with bidirectional, but potentially short-lived, communication links. Inspired by convex recurrence rules for multi-agent systems, and the Metropolis average consensus rule in particular, we designed a deterministic distributed algorithm that achieves asymptotic average consensus, which we show to operate in polynomial time in a synchronous temporal model. The algorithm is easy to implement, has low space and computational complexity, and is fully distributed, requiring neither symmetry-breaking devices like unique identifiers, nor global control or knowledge of the network. In the fully decentralized model that we adopt, to our knowledge, no other distributed average consensus algorithm has a better temporal complexity. Our approach distinguishes itself from classical convex recurrence rules in that the agent's values may sometimes leave their previous convex hull. As a consequence, our convergence bound requires a subtle analysis, despite the syntactic simplicity of our algorithm.

This work was published in [12].

8.9 Modeling

A Kappa model for hepatic stellate cells activation by TGFB1

Participants: Matthieu Bougéon, Pierre Boutillier, Jérôme Feret, Octave Hazard, Nathalie Théret.

In this [16], we model as a realistic case study, a population of hepatic stellate cells under the effect of the TGFB1 protein. In this case study, the components will be occurrences of hepatic stellate cells in different states, and occurrences of the protein TGFB1. The protein TGFB1 induces different behaviors of hepatic stellate cells thereby contributing either to tissue repair or to fibrosis. Better understanding the overall behavior of the mechanisms that are involved in these processes is a key issue to identify markers and therapeutic targets likely to promote the resolution of fibrosis at the expense of its progression.

8.10 Static analysis of signaling pathways

Static analysis for rule-based models

Participants: Jérôme Feret.

In the context biochemical systems, in the first steps of modeling, static analysis helps the modeler by warning about potential issues in the model. Then it provides useful properties to check that what is implemented is what the modeler has in mind and to provide a quick overview of the model for the people who have not written it. In the chapter [17], we recall the basic ingredients of the language Kappa and we explain how local patterns can be used as a cornerstone to build extensible static analyses.

Rate Equations for Graphs

Participants: Vincent Danos, Tobias Heindel, Ricardo Honorato-Zimmer, Sandro Stucki.

We combine ideas from: 1) graph transformation systems (GTSs) stemming from the theory of formal languages and concurrency, and 2) mean field approximations (MFAs), a collection of approximation techniques ubiquitous in the study of complex dynamics to build a framework which generates rate equations for stochastic GTSs and from which one can derive MFAs of any order (no longer limited to the humanly computable). The procedure for deriving rate equations and their approximations can be automated. An implementation and example models are available online. We apply our techniques and tools to derive an expression for the mean velocity of a two-legged walker protein on DNA.

8.11 Formal relationships between modeling paradigms

A generic framework to coarse-grain stochastic reaction networks by Abstract Interpretation

Participants: Jérôme Feret, Albin Salazar.

In the last decades, logical or discrete models have emerged as a successful paradigm for capturing and predicting the behaviors of systems of molecular interactions. Intuitively, they consist in sampling the abundance of each kind of biochemical entity within finite sets of intervals and deriving transitions accordingly. On one hand, formallyproven sound derivation from more precise descriptions (such as from reaction networks) may include many fictitious behaviors. On the other hand, direct modeling usually favors dominant interactions with no guarantee on the behaviors that are neglected. In this paper [13], we formalize a sound coarse-graining approach for stochastic reaction networks. Its originality relies on two main ingredients. Firstly, we abstract values by intervals that overlap in order to introduce a minimal effort for the system to go back to the previous interval, hence limiting fictitious oscillations in the coarse-grained models. Secondly, we compute for pairs of transitions (in the coarse-grained model) bounds on the probabilities on which one will occur first. We illustrate our ideas on two case studies and demonstrate how techniques from Abstract Interpretation can be used to design more precise discretization methods, while providing a framework to further investigate the underlying structure of logical and discrete models.

9 Bilateral contracts and grants with industry

9.1 Bilateral contracts with industry

9.1.1 Disco project with Tezos

Participants: Bernadette Charron-Bost, Cezara Drăgoi, Jérôme Feret, Xavier Rival.

- Title: DISCO: Synchronous Abstractions for Blockchain Infrastructures
- Type: Research contracts funded by Tezos
- Duration: September 2020 - September 2023
- Inria contact: Xavier Rival, Jérôme Feret
- Abstract: The literature in distributed computing distinguishes two main classes of computational models: asynchronous models have better performance, whereas synchronous models provide stronger formal guarantees. Implementations of distributed systems must operate in asynchronous models of computation, where performance emerges from the load of the system. The correctness of asynchronous protocols is very hard to prove, due to the challenges of concurrency, faults, buffered message queues, and message loss, altering, and re-ordering by the network. In contrast, synchronous models are based on (communication- closed) rounds, and this structure greatly facilitates verification. There are no interleavings, and the cumulative size of reception buffers is bounded by the number of processes in the network.

  The goal of this project is to increase the confidence we have in blockchain systems. We propose to: (1) define a synchronous computational model for blockchain algorithms and build a domain-specific language appropriate for this synchronous computational model, (2) equip the domain-specific language with support for mechanized formal verification with a high degree of automation, and (3) prototypically implement a dedicated runtime for efficiently executing, within an asynchronous context, algorithms defined for a synchronous models, together with a formal correctness proof that certifies the correctness of the synchronous abstraction with respect to the asynchronous runtime.

9.1.2 Collaboration with Fujitsu on static analysis for machine learning

Participants: Caterina Urban.

- Title: Formal Verification Techniques for Machine Learning Systems
- Type: Research contract funded by Fujitsu
- Duration: October 2021 - September 2022
- Inria contact: Caterina Urban
- Abstract: The goal of this project is to develop formal verification techniques based on static analysis to verify input-output safety properties of neural network-based classifiers (e.g., local robustness to adversarial perturbations). Specifically, we would like to investigate techniques that are able to quantify and specify the conditions under which a safety property is violated (e.g., which pixels values of an image are susceptible of impacting the classification of the image). To this end, we envision developing a combination of an over-approximating analysis that will efficiently partition the input space, and an under-approximating analysis that will efficiently identify the partitions that violate the given property of interest. We hope that the techniques developed during this collaboration can accompany the deployment of real-world machine learning models by ensuring their safety and by guiding the development of countermeasures to counteract safety issues (e.g., adding a "harness" that ensures that the right output is returned by a neural network even for partitions of the input space that violate a given property of interest).

10 Partnerships and cooperations

10.1 National initiatives

10.1.1 DCore

Participants: Jérôme Feret, Gregor Gössler, Jean Krivine, Ivan Lanese, Claudio Antares Mezzina, Davide Sangiorgi, Jean-Bernard Stefani, German Vidál, Gianluigi Zavattaro.

- Title: DCore - Causal Debugging for Concurrent Systems
- Type: ANR générique 2018
- Defi: Société de l'information et de la communication
- Instrument: ANR grant
- Duration: March 2019 - February 2023
- Coordinator: INRIA Grenoble - Rhône-Alpes (France)
- Others partners: IRIF (France), Inria Paris (France)
- Inria contact: Jérôme Feret
- Abstract: As software takes over more and more functionalities in embedded and safety-critical systems, bugs may endanger the safety of human beings and of the environment, or entail heavy financial losses. In spite of the development of verification and testing techniques, debugging still plays a crucial part in the arsenal of the software developer. Unfortunately, usual debugging techniques do not scale to large concurrent and distributed systems: they fail to provide precise

and efficient means to inspect and analyze large concurrent executions; they do not provide means to automatically reveal software faults that constitute actual causes for errors; and they do not provide succinct and relevant explanations linking causes (software bugs) to their effects (errors observed during execution).

The overall objective of the project is to develop a semantically well-founded, novel form of concurrent debugging, which we call "causal debugging", that aims to alleviate the deficiencies of current debugging techniques for large concurrent software systems.

Briefly, the causal debugging technology developed by the DCore project will comprise and integrate two main novel engines:

1. A reversible execution engine that allows programmers to backtrack and replay a concurrent or distributed program execution, in a way that is both precise and efficient (only the exact threads involved by a return to a target anterior or posterior program state are impacted);
2. a causal analysis engine that allows programmers to analyze concurrent executions, by asking questions of the form "what caused the violation of this program property?", and that allows for the precise and efficient investigation of past and potential program executions.

The project will build its causal debugging technology on results obtained by members of the team, as part of the past ANR project REVER, on the causal semantics of concurrent languages, and the semantics of concurrent reversible languages, as well as on recent works by members of the project on abstract interpretation, causal explanations and counterfactual causal analysis.

The project primarily targets multithreaded, multicore and multiprocessor software systems, and functional software errors, that is errors that arise in concurrent executions because of faults (bugs) in software that prevents it to meet its intended function. Distributed systems, which can be impacted by network failures and remote site failures are not an immediate target for DCore, although the technology developed by the project should constitute an important contribution towards full-fledged distributed debugging. Likewise, we do not target performance or security errors, which come with specific issues and require different levels of instrumentation, although the DCore technology should prove a key contribution in these areas as well.

10.1.2 REPAS

The project REPAS, Reliable and Privacy-Aware Software Systems via Bisimulation Metrics (coordination Catuscia Palamidessi, INRIA Saclay), aims at investigating quantitative notions and tools for proving program correctness and protecting privacy, focusing on bisimulation metrics, the natural extension of bisimulation on quantitative systems. A key application is to develop mechanisms to protect the privacy of users when their location traces are collected. Partners: Inria (Comete, Focus), ENS Cachan, ENS Lyon, University of Bologna.

10.1.3 SAFTA

- Title: SAFTA Static Analysis for Fault-Tolerant distributed Algorithms.
- Type: ANR JCJC 2018
- Duration: February 2018 - August 2022
- Coordinator: Cezara Drăgoi, CR Inria
- Abstract: Fault-tolerant distributed data structures are at the core distributed systems. Due to the multiple sources of non-determinism, their development is challenging. The project aims to increase the confidence we have in distributed implementations of data structures. We think that the difficulty does not only come from the algorithms but from the way we think about distributed systems. In this project we investigate partially synchronous communication-closed round based programming abstractions that reduce the number of interleavings, simplifying the reasoning about distributed systems and their proof arguments. We use partial synchrony to define reduction

theorems from asynchronous semantics to partially synchronous ones, enabling the transfer of proofs from the synchronous world to the asynchronous one. Moreover, we define a domain specific language, that allows the programmer to focus on the algorithm task, it compiles into efficient asynchronous code, and it is equipped with automated verification engines.

10.1.4 VERIAMOS

- Title: Verification of Abstract Machines for Operating Systems
- Type: ANR générique 2018
- Defi: Société de l'information et de la communication
- Instrument: ANR grant
- Duration: January 2019 - December 2022
- Coordinator: INRIA Paris (France)
- Others partners: LIP6 (France), IRISA (France), UGA (France)
- Inria contact: Xavier Rival
- Abstract: Operating System (OS) programming is notoriously difficult and error prone. Moreover, OS bugs can have a serious impact on the functioning of computer systems. Yet, the verification of OSes is still mostly an open problem, and has only been done using user-assisted approaches that require a huge amount of human intervention. The VeriAMOS proposal relies on a novel approach to automatically and fully verifying OS services, that combines Domain Specific Languages (DSLs) and automatic static analysis. In this approach, DSLs provide language abstraction and let users express complex policies in high-level simple code. This code is later compiled into low level C code, to be executed on an abstract machine. Last, the automatic static analysis verifies structural and robustness properties on the abstract machine and generated code. We will apply this approach to the automatic, full verification of input/output schedulers for modern supports like SSDs.

11 Dissemination

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

11.1 Promoting scientific activities

11.1.1 Scientific events: organisation

General chair, scientific chair

- Jérôme Feret is a member of the Steering Committee of the Workshop on Static Analysis and Systems Biology (SASB).
- Caterina Urban is a member of the Steering Committee of the International Workshop on the State Of the Art in Program Analysis (SOAP).
- Caterina Urban is a member of the ETAPS Executive Board.

Member of the organizing committees

- Caterina Urban is a member of the organizing committee of the Mentoring Workshop at ETAPS 2023.
- Caterina Urban was a member of the organizing committee of the Mentoring Workshop at FLoC 2022.
- Caterina Urban was a member of the organizing committee of the Mentoring Workshop at ETAPS 2022.

11.1.2 Scientific events: selection

Chair of conference program committees

- Caterina Urban is chairing the committee of the ETAPS Doctoral Dissertation Award 2023.
- Caterina Urban chaired the Program Committee of the 29th Static Analysis Symposium (SAS 2022).
- Caterina Urban chaired the Student Research Competition of SPLASH 2022.
- Caterina Urban chaired the Posters of SPLASH 2022.
- Caterina Urban chaired the committee of the ETAPS Doctoral Dissertation Award 2022.

Member of the conference program committees

- Jérôme Feret served as a Member of the Program Committee of CMSB 2022 (Conference on Computational Methods in Systems Biology).
- Jérôme Feret served as a Member of the Program Committee of SAS 2022 (Static Analysis Symposium).
- Jérôme Feret served as a Member of the Program Committee of VMCAI 2023 (Verification, Model Checking, and Abstract Interpretation).
- Xavier Rival served as a Member of the Program Committee of CAV 2022 (Computer-Aided Verification).
- Xavier Rival served as a Member of the Program Committee of VMCAI 2022 (Verification, Model Checking and Abstract Interpretation).
- Caterina Urban is serving as a Member of the Program Committee of TACAS 2023 (Tools and Algorithms for the Construction and Analysis of Systems).
- Caterina Urban is serving as a Member of the Program Committeee of CAV 2023 (Computer-Aided Verification).
- Caterina Urban is serving as a Member of the Program Committee of NFM 2023 (NASA Formal Methods).
- Caterina Urban served as a Member of the Program Committee of ESOP 2023 (European Symposium on Programming).
- Caterina Urban served as a Member of the Program Committee of ICTAC 2022 (International Colloquium on Theoretical Aspects of Computing).
- Caterina Urban served as a Member of the Program Committee of CAV 2022 (Computer-Aided Verification).
- Caterina Urban served as a Member of the Program Committee of POPL 2022 (Principles of Programming Languages).
- Caterina Urban served as a Member of the Reviewing Committee of the ACM Student Research Competition 2022.

Reviewer

- Xavier Rival served as a Reviewer for POPL 2023.
- Caterina Urban served as a Reviewer for OOPSLA 2022 (Object-oriented Programming, Systems, Languages, and Applications).
- Caterina Urban served as a Reviewer for FMCAD 2022 (Formal Methods in Computer-Aided Design).

11.1.3 Journal

Member of the editorial boards

- Caterina Urban is serving as a Guest Editor for the Special Issue on SAS 2022 of Formal Methods in System Design.
- Caterina Urban is serving as a Guest Editor for the Special Issue on CAV 2020 of Formal Methods in System Design.

Reviewer - reviewing activities

- Jérôme Feret served as a Reviewer for MSCS (Mathematical Strucutres in Computer Science).
- Jérôme Feret served as a Reviewer for FMSD (Formal Methods in System Design).
- Jérôme Feret served as a Reviewer for SCP (Science of Computer Programming).
- Jérôme Feret served as a Reviewer for PeerJ.
- Jérôme Feret served as a Reviewer for BMC Supplements.
- Xavier Rival served as a Reviewer for PACML-POPL 2023 (Proceedings of the ACM, Principles of Programming Languages).
- Xavier Rival served as a Reviewer for FMSD (Formal Methods in System Design).
- Caterina Urban served as a Reviewer for CACM (Communications of the ACM).
- Caterina Urban served as a Reviewer for FMSD (Formal Methods in System Design).
- Caterina Urban served as a Reviewer for TOPLAS (Transactions on Programming Languages and Systems).

11.1.4 Invited talks

- Jérôme Feret gave an invited talk at the National Days of the GDR IM (Informatique Mathématique).
- Xavier Rival gave an invited talk at the IRIF Seminar in January 2022.
- Xavier Rival gave an invited talk at the Probabilistic Programming Workshop organised at College de France in July 2022.
- Xavier Rival gave an invited talk at KAIST (South Korea) in October 2022.
- Xavier Rival gave an invited talk at Seoul National University (South Korea) in November 2022.
- Xavier Rival was invited to give a session preview lecture at POPL 2023.
- Caterina Urban will give an invited talk at the International Symposium on Model Checking of Software (SPIN 2023).
- Caterina Urban will give an invited talk at the Séminaire IRILL of the Center for Research and Innovation on Free Software.

- Caterina Urban gave an invited talk at the Dagstuhl Seminar 22291 “Machine Learning and Logical Reasoning: The New Frontier”.
- Caterina Urban gave an invited talk at the “From Theory to Practice” Workshop within the Verified Software Programme of Isaac Newton Institute for Mathematical Sciences.
- Caterina Urban gave an invited talk at the International Workshop on the State Of the Art in Program Analysis (SOAP 2022).
- Caterina Urban gave an invited talk at the "Challenges of Software Verification” Workshop of the Ca’ Foscari University.

11.1.5 Leadership within the scientific community

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

11.1.6 Research administration

- Jérôme Feret is a Member of the Laboratory Council of the Department of Computer Science of École normale supérieure.
- 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.
- Xavier Rival is a Member of the Laboratory Council of the Department of Computer Science of École normale supérieure.
- Xavier Rival is a Member of the Evaluation Committee of INRIA.

11.2 Teaching - Supervision - Juries

11.2.1 Teaching

- Licence:
  - Jérôme Feret and Xavier Rival (lectures), and Josselin Giet (tutorials), “Semantics and Application to Verification”, 36h, L3, at École Normale Supérieure, France.
- Master:
  - Bernadette Charron-Bost, ”Calculability in multi-agent networks”, 24h, M2, Parisian Master of Research in Computer Science (MPRI), France.
  - Bernadette Charron-Bost, ”Consensus problems”, 24h, M1 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.
  - Jérôme Feret, Jean Krivine, Sébastien Légaré, and Matthieu Bouguéon. Rule-based Modelling, 24h, M1. Interdisciplinary Approaches to Life Science (AIV), Master Program, Université Paris-Descartes, France.

11.2.2 Supervision

- Internship: Bernadette Charron-Bost supervised the M1 internship of Guillaume Prémel, from April 2022 to September 2022.
- Internship: Josselin Giet and Xavier Rival supervised the internship of Felix Ridoux from ENS Rennes, from September 2022 to December 2022, on the static analysis of binary search tree data-structures.
- Internship: Caterina Urban supervised the Ecole Polytechnique L3 internship of Guruprerana Shabadi, on “Static Analysis of Jupyter Notebooks”.
- Internship: Caterina Urban supervised the PhD internship of Luca Negrini (Luca Negrini is PhD student at Università Ca’ Foscari Venezia), from January to April 2022, on “Static Analysis of Jupyter Notebooks in PyLisa”.
- Internship: Caterina Urban supervised the ENS Paris Saclay M2 internship of “Data Expectations Static Analysis for Jupyter Notebooks” from March 2022 to August 2022.
- Internship: Caterina Urban supervised the Bachelor internship of Abhinandan Pal (IIT Kalyani), from November 2022 to January 2023, on “Attention Robustness Static Analysis”.
- Internship: Caterina Urban supervised the ENS Rennes M2 internship of Orphée Radet, from December 2022 to January 2023, on “Fairness Static Analysis for Jupyter".
- PhD in progress: Patricio Inzaghi, A Sequentialization Procedure for Fault-Tolerant Protocols, started in September 2019 and supervised by Cezara Dragoi.
- PhD in progress: Louis Penet de Monterno, Synchronization in multi-agent networks, started in September 2020 and supervised by Bernadette Charron-Bost.
- 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: 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.
- PhD in progress: Albin Salazar, Formal derivation of discrete models with separated time-scales, started in 2019 and supervised by Jérôme Feret.
- PhD in progress: Josselin Giet, Static Analysis of components of operating systems by abstract interpretation, started in 2020 and supervised by Xavier Rival.
- PhD in progress: Ignacio Tiraboschi, Static analysis of security properties for IoT systems, started in 2020 and co-supervised by Tamara Rezk (EP Indes) and Xavier Rival.
- PhD in progress: Serge Durand, Formal Specification of Machine Learning Algorithms, started in 2021 and supervised by Zakaria Chihani (CEA/List) and Caterina Urban.
- PhD in progress: Denis Mazzucato, Static Analysis by Abstract Interpretation of Machine-Learned Software, started in 2020 and supervised by Caterina Urban.

11.2.3 Juries

- Jérôme Feret served as a member of the Review Committee for the PhD of Julien Braine at École normale supérieure of Lyon (Defense: May 2022).
- Jérôme Feret was Reviewer for the PhD of Samuel Pastva at Masaryk University (Brno, Czech Republic, Defense: June 2022).

- Jérôme Feret served as a member of the Review Committee for the PhD of David Delmas at Sorbone University (Defense: December 2022).
- Xavier Rival chaired the PhD defense committee of Basile Clément (ENS, Defense: September 2022).
- Xavier Rival served as a Jury member for the defense of the PhD of Adam Khayam (University of Rennes, Defense: November 2022).
- Xavier Rival is a reviewer for the Habilitation Thesis Committee of Arthur Charguéraud (University of Strasbourg).
- Caterina Urban served as a member of the Review Committee for the PhD of Guillaume Eric Vidot at Université Toulouse 2 - Jean Jaurès (Defense: December 2022).
- Caterina Urban served as a member of the Review Committee for the PhD of Guillaume Girol at CEA and Université Paris-Saclay (Defense: October 2022).

11.3 Popularization

11.3.1 Internal or external Inria responsibilities

- Jérôme Feret served in the “admissibility” jury for INRIA researcher positions (CRCN) for the center of “Paris-Saclay” in 2022.
- Jérôme Feret chaired the recrutting committee for a position of teaching assistant at University of Evry.
- Xavier Rival is a member of the Bureau du Comité des Projets.
- Xavier Rival is serving as a member of the Evaluation Committee of INRIA (CE).
- Xavier Rival served in the “admissibility” jury for INRIA researcher positions (CRCN) for the center of “Paris-Saclay” in 2022.
- Xavier Rival served in the “admissibility” jury for INRIA senior researcher positions (DR2) in 2022.
- Xavier Rival served in the HCERES Evaluation Committee of the LORIA Laboratory (Nancy) in September 2022.
- Caterina Urban is serving as a member of the assessment committee for a position of associate professor in Systems and Software Engineering at University of Copenhagen.
- Caterina Urban served as a member of the Jury d’Admissibilité CRCN/ISFP at Inria Paris.
- Caterina Urban served as a member of the Commission des Emplois Scientifiques at Inria Paris.

11.3.2 Education

- Caterina Urban was a lecturer in the 4th International Programming Language Implementation Summer School (PLISS 2022) in Bertinoro, Italy.
- Caterina Urban was a lecturer in the Advanced Track of the 13th International School of Rewriting (ISR 2022)in Tbilisi, Georgia.
- Caterina Urban was a lecturer in the 2nd Inria-DFKI European Summer School on Artificial Intelligence (IDESSAI 2022) in Saarbrücken, Germany.

11.3.3 Interventions

- Caterina Urban gave a talk on “Interprétation Abstraite des Réseaux de Neurones” for La Demi-Heure de Science of Inria Paris.

12 Scientific production

12.1 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, C. Radhia, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival. 'A Static Analyzer for Large Safety-Critical Software'. In: *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI'03)*. ACM Press, June 2003, pp. 196–207.

[3] A. Bouajjani, C. Dragoi, C. Enea and M. Sighireanu. 'On inter-procedural analysis of programs with lists and data'. In: *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*. 2011, pp. 578–589. DOI: 10.1145/1993498.1993566. URL: http://doi.acm.org/10.1145/1993498.1993566.

[4] 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.

[5] J. Feret, V. Danos, J. Krivine, R. Harmer and W. Fontana. 'Internal coarse-graining of molecular systems'. In: *Proceeding of the national academy of sciences* 106.16 (Apr. 2009).

[6] 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.

[7] A. Miné. 'The Octagon Abstract Domain'. In: *Higher-Order and Symbolic Computation* 19 (2006), pp. 31–100.

[8] 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.

12.2 Publications of the year

International journals

[9] B. Charron-Bost. 'Geometric bounds for convergence rates of averaging algorithms'. In: *Information and Computation* 285 (May 2022), p. 104909. DOI: 10.1016/j.ic.2022.104909. URL: https://hal.science/hal-03717768.

[10] W. Lee, X. Rival and H. Yang. 'Smoothness Analysis for Probabilistic Programs with Application to Optimised Variational Inference'. In: *Proceedings of the ACM on Programming Languages* 7 (15th Jan. 2023), pp. 335–366. DOI: 10.1145/3571205. URL: https://hal.science/hal-03936759.

[11] W. Waites, M. Cavaliere, V. Danos, R. Datta, R. M. Eggo, T. B. Hallett, D. Manheim, J. Panovska-Griffiths, T. W. Russell and V. I. Zarnitsyna. 'Compositional modelling of immune response and virus transmission dynamics'. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 380.2233 (3rd Oct. 2022). DOI: 10.1098/rsta.2021.0307. URL: https://hal.science/hal-03423441.

International peer-reviewed conferences

[12] B. Charron-Bost and P. Lambein-Monette. 'Computing Outside the Box: Average Consensus over Dynamic Networks'. In: 1st Symposium on Algorithmic Foundations of Dynamic Networks, SAND 2022. Vol. 221. LIPIcs. Roma, Italy: Schloss Dagstuhl - Leibniz-Zentrum fur Informatik, 28th Mar. 2022. DOI: 10.4230/LIPIcs.SAND.2022.10. URL: https://hal.science/hal-03717753.

[13] J. Feret and A. Salazar. 'A generic framework to coarse-grain stochastic reaction networks by Abstract Interpretation'. In: VMCAI 2023 - 24th International Conference on Verification, Model Checking and Abstract Interpretation. Boston, United States, 15th Jan. 2023. URL: https://hal.inria.fr/hal-03886237.

[14] O. Nicole, M. Lemerre and X. Rival. 'Lightweight Shape Analysis based on Physical Types'. In: VMCAI 2022 - 23rd International Conference on Verification, Model Checking, and Abstract Interpretation. Philadelphia, United States, 16th Jan. 2022. URL: https://hal.science/hal-03538088.

[15] I. Tiraboschi, T. Rezk and X. Rival. 'Sound Symbolic Execution via Abstract Interpretation and its Application to Security'. In: *Lecture Notes in Computer Science*. 24th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2023). Vol. 13881. Verification, Model Checking, and Abstract Interpretation 24th International Conference, VMCAI 2023, Boston, MA, USA, January 16–17, 2023, Proceeding. Boston, MA, United States: Springer Nature Switzerland, 17th Jan. 2023, pp. 267–295. DOI: 10.1007/978-3-031-24950-1_13. URL: https://hal.science/hal-03942146.

Scientific book chapters

[16] M. Bouguéon, P. Boutillier, J. Feret, O. Hazard and N. Théret. 'The rule-based model approach. A Kappa model for hepatic stellate cells activation by TGFB1'. In: *Systems Biology Modelling and Analysis: Formal Bioinformatics Methods and Tools*. Wiley, Nov. 2022, pp. 1–76. URL: https://hal.inria.fr/hal-03388100.

[17] J. Feret. 'Analyses des motifs accessiblesdans les modèles Kappa'. In: *Approches symboliques de la modélisation et de l'analyse des systèmes biologiques*. ISTE, July 2022. URL: https://hal.inria.fr/hal-03088539.

Doctoral dissertations and habilitation theses

[18] O. Nicole. 'Automated verification of systems code using type-based memory abstractions'. École normale supérieure - PSL; CEA List, 19th Apr. 2022. URL: https://theses.hal.science/tel-03962643.

Reports & preprints

[19] F. Drobnjaković, P. Subotić and C. Urban. *Abstract Interpretation-Based Data Leakage Static Analysis*. Microsoft Research; Inria Paris; École Normale Supérieure, 29th Nov. 2022. URL: https://hal.inria.fr/hal-03926245.

[20] S. Munakata, C. Urban, H. Yokoyama, K. Yamamoto and K. Munakata. *Verifying Attention Robustness of Deep Neural Networks against Semantic Perturbations*. Fujitsu; Inria Paris; École Normale Supérieure, 13th July 2022. URL: https://hal.inria.fr/hal-03926254.

[21] A. Pal, F. Ranzato, C. Urban and M. Zanella. *Abstract Interpretation-Based Feature Importance for SVMs*. IIIT Kalyani; University of Padova; Inria Paris; École Normale Supérieure, 22nd Oct. 2022. URL: https://hal.inria.fr/hal-03926237.

Other scientific publications

[22] S. Durand, A. Lemesle, Z. Chihani, C. Urban and F. Terrier. 'ReCIPH: Relational Coefficients for Input Partitioning Heuristic'. In: 1st Workshop on Formal Verification of Machine Learning (WFVML 2022). Baltimore, United States, 22nd July 2022. URL: https://hal.inria.fr/hal-03926281.

[23] S. Munakata, C. Urban, H. Yokoyama, K. Yamamoto and K. Munakata. 'Verifying Attention Robustness of Deep Neural Networks against Semantic Perturbations'. In: 29th Asia-Pacific Software Engineering Conference (APSEC 2022). [Virtual], Japan, 6th Dec. 2022. URL: https://hal.inria.fr/hal-03926269.

12.3 Cited publications

[24] W. Lee, H. Yu, X. Rival and H. Yang. 'Towards Verified Stochastic Variational Inference for Probabilistic Programs'. In: *Proceedings of the ACM on Programming Languages* 16 (2020). DOI: 10.1145/3371084. URL: https://hal.archives-ouvertes.fr/hal-02399922.

[25] 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. URL: http://dx.doi.org/10.1016/S1571-0661(05)80168-9.

[26] 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.

[27] C. Dragoi, C. Enea, B. K. Ozkan, R. Majumdar and F. Niksic. ‘Testing consensus implementations using communication closure’. In: *SPLASH 2020 : ACM SIGPLAN conference on Systems, Programming, Languages, and Applications: Software for Humanity*. Chiccago / Virtual, United States, Oct. 2021. DOI: 10.1145/3428278. URL: https://hal.inria.fr/hal-03134294.

[28] O. Nicole, M. Lemerre, S. Bardin and X. Rival. ‘No Crash, No Exploit: Automated Verification of Embedded Kernels’. In: *RTAAS 2021 - Real-Time and Embedded Technology and Applications Symposium*. Nashville, United States, May 2021. URL: https://hal.science/hal-03538067.

[29] 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. URL: https://hal.inria.fr/hal-02397699.