RESEARCH CENTRE

Paris

2021 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

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]
- Vincent Danos [CNRS, Senior Researcher, HDR]
- Jérôme Feret [Inria, Researcher]
- Caterina Urban [Inria, Researcher]

Post-Doctoral Fellows

- Hamza El Khalloufi [Inria, until Sep 2021]
- Adrien Husson [Inria, until May 2021]
- Nikolaos Lamprou [Inria, from Apr 2021 until Sep 2021]

PhD Students

- Josselin Giet [École Normale Supérieure de Paris]
- Patricio Inzaghi [Inria]
- Denis Mazzucato [Inria]
- Olivier Nicole [CEA]
- Albin Salazar [Inria]
- Ignacio Tiraboschi [Inria]

Technical Staff

- Tie Cheng [Inria, Engineer, until Feb 2021]
- Sebastien Légaré [Inria, Engineer, until Aug 2021]
- Thierry Martinez [Inria, Engineer]

Interns and Apprentices

- Charles De Haro [École normale supérieure de Rennes, from May 2021 until Jul 2021]

Administrative Assistant

- Nathalie Gaudechoux [Inria]

External Collaborator

- Pierre Boutillier [Self-employed]

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* [27]. 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* [27], 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 [26], 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 Staticanalysis 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

Bernadette Charron-Bost joined the ANTIQUE team.

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.

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 [13].

8.2 Relational Static Analysis

Relational abstraction for memory properties.

Participants: Hugo Illous, Matthieu Lemerre, Xavier Rival *(correspondant)*.

Static analyses aim at inferring semantic properties of programs. We can distinguish two important classes of static analyses: state analyses and relational analyses. While state analyses aim at computing an over-approximation of reachable states of programs, relational analyses aim at computing functional

properties over the input-output states of programs. Several advantages of relational analyses are their ability to analyze incomplete programs, such as libraries or classes, but also to make the analysis modular, using input-output relations as composable summaries for procedures. In the case of numerical programs, several analyses have been proposed that utilize relational numerical abstract domains to describe relations. On the other hand, designing abstractions for relations over input-output memory states and taking shapes into account is challenging. We have proposed a set of novel logical connectives to describe such relations, which are inspired by separation logic. This logic can express that certain memory areas are unchanged, freshly allocated, or freed, or that only part of the memory was modified. Using these connectives, we have built an abstract domain and design a static analysis that over-approximates relations over memory states containing inductive structures. We implemented this analysis and evaluated it on a basic library of list manipulating functions.

This work was initially done as part of the PhD of Hugo Illous [28]. Since then, it has been extended and this extension was published in the Journal for Formal Methods in Systems Design (FMSD) in 2021 [9].

Interprocedural Shape Analysis Using Separation Logic-based Transformer Summaries

Participants: Hugo Illous, Matthieu Lemerre, Xavier Rival *(correspondant)*.

Interprocedural static analysis focuses on the analysis of programs with functions, and traditionally relies on two main approaches: the first uses a state abstraction and computes over-approximations for sets of states in a finite collection of abstract contexts; the second abstracts the effect of each procedure using a relation.

Shape analyses aim at inferring semantic invariants related to the data-structures that programs manipulate. To achieve that, they typically abstract the set of reachable states, which implies that they fit nicely with the first approach to interprocedural analysis, but not to the second.

By contrast, abstractions for transformation relations between input states and output states not only provide a finer description of program executions but also enable the composition of the effect of program fragments so as to make the analysis modular. However, few logics can efficiently capture such transformation relations. In this work, we proposed to use connectors inspired by separation logic to describe memory state transformations and to represent procedure summaries. Based on this abstraction, we designed a top-down interprocedural analysis using shape transformation relations as procedure summaries. Finally, we report on implementation and evaluation.

This work was initially done as part of the PhD of Hugo Illous [28] and published at SAS 2020 [23]. We have extended the presentation of this work towards a journal or book chapter publication.

8.3 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 [12].

8.4 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 [21].

8.5 Static Analysis of Neural Networks

Perfectly Parallel Fairness Certification

Participants: Caterina Urban *(correspondant)*, Maria Christakis, Valentin Wüestholz, Fuyuan Zhang.

Recently, there is growing concern that machine-learning models, which currently assist or even automate decision making, reproduce, and in the worst case reinforce, bias of the training data. The development of tools and techniques for certifying fairness of these models or describing their biased behavior is, therefore, critical.

In [22], we propose a perfectly parallel static analysis for certifying causal fairness of feed-forward neural networks used for classification tasks. When certification succeeds, our approach provides definite guarantees, otherwise, it describes and quantifies the biased behavior. We design the analysis to be sound, in practice also exact, and configurable in terms of scalability and precision, thereby enabling pay-as-you-go certification. We implement our approach in an open-source tool and demonstrate its effectiveness on models trained with popular datasets.

8.6 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 [10] 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.

8.7 Modeling

A Kappa model for hepatic stellate cells activation by TGFB1

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

In this [15], 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.

8.8 Causal analysis for signaling pathways

Pathways summary

Participants: Sébastien Légaré, Jean Krivine, Jérôme Feret.

Causality analysis of rule-based models allows the reconstruction of the causal paths leading to chosen events of interest. This potentially reveals emerging paths that were completely unknown at the time of creation of a model. However, current implementations provide results in the form of a collection of stories. For large models, this can amount to hundreds of story graphs to read and interpret for a single event of interest. In [14], we hence develop a method to fold a collection of stories into a single quotient graph.The main challenge is to find a trade-off in the partitioning of story events which will maximize compactness without loosing important details about information propagation inthe model. The partitioning criterion proposed is relevant context, the context from an event's past which remains useful in its future. Each step of the method is illustrated on a toy rule-based model. This work is part of a longer term objective to automatically extract biological pathways from rule-based models.

8.9 Static analysis of signaling pathways

Static analysis

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 [25], 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.

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 Exploratory collaboration with Airbus on static analysis for machine learning

Participants: Caterina Urban.

- Title: Formal Methods for Artificial Intelligence: State of the Art
- Type: Research contract funded by AirBus France
- Duration: October 2020 - December 2020
- Inria contact: Caterina Urban
- Abstract: Artificial intelligence is a key enabler for the development of autonomous aircrafts. In order to use this technology in critical systems, strict safety guarantees are necessary for the trained machine learning models. The actual state of the art in artificial intelligence does not allow providing such guarantees and new methods are currently being developed. Among these, formal methods and notably static analysis by abstract interpretation appear to be the most promising for critical systems, in terms of soundness and scalability. Moreover, formal methods are actually intensively used for the verification of critical avionics software and well accepted by certification authorities. Nevertheless, as many research teams are developing multiple methods that fall under the formal methods umbrella, a need has emerged for Airbus to better understand this academics ecosystem. The goal of this contract is to carry out a thorough report on the state of the art in formal methods for artificial intelligence and discuss perspectives and expectations for possible worthwhile future research directions.

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
- See also: https://project.inria.fr/dcore/
- 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.

10.2 Regional initiatives

Bernadette Charron-Bost is among the principal investigators of the X-Capgemini Chair on blockchain techniques. She is part of the Steering Committee of this Chair.

11 Dissemination

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).
- Xavier Rival is a member of the Steering Committee of the Static Analysis Symposium (SAS).
- Xavier Rival is a member of the Steering Committee of the Workshop on Tools for Automatic Program Analysis (TAPAS).
- Caterina Urban is a member of the Executive Board of ETAPS (European Joint Conferences on Theory & Practice of Software).
- Caterina Urban is a member of the Steering Committee of the Workshop on the State of the Art in Program Analysis (SOAP).

Member of the organizing committees

- Caterina Urban served as Chair of the Award Committee of the ETAPS Doctoral Dissertation Award 2021.
- Caterina Urban organized the 7th Verification Mentoring Workshop @CAV 2021.
- Caterina Urban is serving as Chair of the Award Committee of the ETAPS Doctoral Dissertation Award 2022.
- Caterina Urban is organizing the Mentoring Workshop @ETAPS 2022.
- Caterina Urban is organizing the Mentoring Workshop @LICS 2022.

11.1.2 Scientific events: selection

Chair of conference program committees

- Caterina Urban served as Chair of SOAP 2021 (Workshop on the State of the Art in Program Analysis).
- Caterina Urban served as Chair of the SPLASH 2021 Student Research Competition.
- Caterina Urban is serving as Chair of the SPLASH 2022 Student Research Competition.
- Caterina Urban is serving as Chair of SAS 2022 (Static Analysis Symposium).

Member of the conference program committees

- Jérôme Feret is serving as a Member of the Program Committee of CMSB 2021 (Conference on Computational Methods in Systems Biology).
- Jérôme Feret is serving as a Member of the Program Committee of SAS 2021 (Static Analysis Symposium) and chaired a session.
- Xavier Rival served as a Member of the Program Committee of SAS 2021 (Static Analysis Symposium).
- Xavier Rival served as a Member of the Program Committee of VMCAI 2021 (Static Analysis Symposium) and chaired a session.
- Caterina Urban served as a member of the Program Committee of FAccT 2021 (Fairness, Accountability, and Transparency)
- Caterina Urban served as a member of the Program Committee of NFM 2021 (NASA Formal Methods)
- Caterina Urban served as a member of the Program Committee of CAV 2021 (Computer Aided Verification)
- Caterina Urban served as a member of the Program Committee of SBLP 2021 (Brazilian Symposium on Programming Languages)
- Caterina Urban served as a member of the Ethical Review Committee of NeurIPS 2021 (Neural Information Processing Systems)
- Caterina Urban served as a member of the Program Committee of POPL 2022 (Symposium on Principles of Programming Languages)
- Caterina Urban is serving as a member of the Program Committee of CAV 2022 (Computer-Aided Verification)
- Caterina Urban is serving as a member of the Program Committee of ICTAC 2022 (Theoretical Aspects of Computing)
- Caterina Urban is serving as a member of the Program Committee of ESOP 2023 (European Symposium on Programming)

Reviewer

- Jérôme Feret served as Reviewer for MFCS 2021 (Symposium on Mathematical Foundations of Computer Science).
- Caterina Urban served as Reviewer for SAS 2021 (Static Analysis Symposium)

11.1.3 Journal

Member of the editorial boards

- Jérôme Feret is in the editorial board of Open Journal of Modelling and Simulation.
- Jérôme Feret is in the editorial board of Frontiers in Genetics.

Guest editors of special issues

- Jérôme Feret served as a guest editor (with Cedric Lhoussaine) of the special issue "Formal Method for Biological Systems Modelling", appeared in the section "Computational Biology" of "Computation". Volume 9. 2021 [17].
- Caterina Urban is serving as a guest editor (with Aws Albarghouthi) of the special issue of Formal Methods in System Design on CAV 2020 (Computer-Aided Verification)

Reviewer - reviewing activities

- Jérôme Feret serves as a reviewer for Theoretical Computer Science.
- Jérôme Feret serves as a reviewer for Transactions on Computational Biology and Bioinformatics.
- Xavier Rival served as a reviewer for Formal Methods in System Design.
- Caterina Urban served as a reviewer for Software: Practice and Experience.
- Caterina Urban served as a reviewer for Transactions on Programming Languages and Systems.

11.1.4 Invited talks

- Xavier Rival gave an invited presentation at the fourth Workshop on Probabilistic Interactive and Higher-Order Computation (PIHOC'21).
- Xavier Rival gave an invited talk at the IRIF seminar in May 2021.
- Caterina Urban gave an invited talk at the Lorentz Center Workshop "Robust Artificial Intelligence" in January 2021.
- Caterina Urban gave an invited talk at Airbus in February 2021.
- Caterina Urban gave an invited talk at the 4th Workshop on Formal Methods for ML-Enabled Autonomous Systems (FoMLAS) in July 2021.
- Caterina Urban gave an invited talk at CEA-LIST in November 2021.
- Caterina Urban gave an invited talk at the Journées du GT Vérif 2021 at ENS Paris-Saclay in November 2021.

11.1.5 Leadership within the scientific community

- Xavier Rival is a member of the IFIP Working Group 2.4 on Software implementation technology.

11.1.6 Scientific expertise

- Jérôme Feret evaluated a project proposal for the FNR Luxembourg CORE programme.
- Jérôme Feret evaluated four project proposals for the PRIN 2021 programme, funded by the Italian Ministry of Education, University and Research.
- Xavier Rival evaluated a project proposal for the FNR Luxembourg CORE programme.
- Xavier Rival reviewed a project for the GV COFUND - University Ca' Foscari of Venice, Italy.
- Caterina Urban evaluated a project proposal for the ERC 2021 programme.

11.1.7 Research administration

- Jérôme Feret and Xavier Rival are members of the Laboratory Council of DIENS.
- Jérôme Feret is 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.
- Bernadette Charron-Bost served as member of the hiring committee at LIX (Ecole Polytechnique).
- Bernadette Charron-Bost is a member of the committee of Administration and scientific committee of the LabEx CIMI.

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:
  - 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, 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.
  - Pierre Boutillier, Jérôme Feret, and Jean Krivine, Rule-based Modelling, 24h, M1. Interdisciplinary Approaches to Life Science (AIV), Master Program, Université Paris-Descartes, France.
- Doctorat:
  - Caterina Urban, “Abstract Interpretation and Applications Beyond the Beaten Track”, 6h, PhD. Doctoral Program in Computer Science at the Gran Sasso Science Institute, Italie.

11.2.2 Supervision

- PhD in progress: Serge Durand, Specification and Formal Verification of Machine Learning Algorithms, started in 2021 and supervised by Caterina Urban and Zakaria Chihani (CEA)
- 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: Denis Mazzucato, Static Analysis by Abstract Interpretation of Machine-Learned Software, started in 2020 and supervised by Caterina Urban.
- 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: Olivier Nicole, Verification of micro-kernels, started in 2018 and supervised by Xavier Rival and Matthieu Lemerre (CEA)
- PhD in progress: Josselin Giet, Functional verification of components of operating systems by static analysis, started in 2020 and supervised by Xavier Rival and Gilles Muller (INRIA Paris, Project team Whisper).
- PhD in progress: Ignacio Tiraboshi, Static analysis for security properties on IoT applications, started in 2020 and supervised by Xavier Rival and Tamara Rezk (INRIA Sophia, Project team Indes).
- PhD in progress: Louis Penet de Monterno, started in October 2020 and supervised by Bernadette Charron-Bost.
- BsC internship: Xavier Rival supervised the internship of Jiwon Park (Bachelor of Ecole Polytechnique, France) on the study in Summer 2021.
- BsC internship: Josselin Giet and Xavier Rival supervised the internship of Charles de Haro (ENS Bretagne) in Summer 2021.

11.2.3 Juries

- Bernadette was reviewer for the PhD of Grégoire Bonin at University of Nantes and served as a member of the Review Committee (Defense: November 2021).
- Bernadette was reviewer for the PhD of Eloi Perdereau at University of Aix-Marseilles and served as a member of the Review Committee (Defense: December 2021).
- Jérôme Feret served as a member of the Review Committee for the PhD of Diane Gallois-Wong at Paris-Saclay University (Defense: March 2021).
- Xavier Rival was reviewer for the PhD of Marco Campion and served as a member of the Review Committee, for the University of Verona (Defense: October 2021).
- Xavier Rival served as a member of the Review Committee for the PhD of Samuele Buro at University of Verona (Defense: October 2021).
- Xavier Rival served as a member of the Review Committee for the PhD of Georges-Axel Jaloyan at ENS Paris (Defense: September 2021).
- Caterina Urban was reviewer for the PhD of Marco Zanella and served as a member of the Review Committee, for the University of Padova (Defense: March 2021).
- Caterina urban served as a member of the Review Committee for the PhD of Julien Girard-Satabin at Université Paris-Saclay (Defense: November 2021).

11.3 Popularization

11.3.1 Internal or external Inria responsibilities

- Jérôme Feret is a member of the jury of the ISIF - Gilles Kahn PhD Award.
- Jérôme Feret served in the “admissibility” jury for INRIA researcher positions (CRCN) for the center of “Paris-Saclay” in 2021.
- Xavier Rival is elected members of the INRIA Commision of Evaluation
- Xavier Rival is member of the “Bureau du comité des projets” of the Paris INRIA Research Center.
- Xavier Rival served in the “admissibility” jury for INRIA senior researcher positions (DR2) in 2021.
- Xavier Rival served in the “admissibility” jury for INRIA researcher positions (CRCN) for the center of “Nancy Grand Est” in 2021.
- Xavier Rival served in the “admission” jury for INRIA senior researcher positions (DR2) in 2021.
- Xavier Rival served in the “admission” jury for INRIA researcher positions (CRCN) for all centers in 2020.
- Caterina Urban served in the INRIA Commission Emplois Scientifique in 2021.
- Caterina Urban is serving in the INRIA Commission Emplois Scientifique in 2022.
- Caterina Urban is serving in the “admission” jury for INRIA researcher positions (CRCN) for all centers in 2022.

11.3.2 Articles and contents

- Xavier Rival published with Kwangkeun Yi a book aimed to serve as an introduction to the field of static analysis, for students and engineers, released by MIT Press in January 2020 [24].

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, R. Cousot, 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\discretionary–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] H. Illous, M. Lemerre and X. Rival. 'A relational shape abstract domain'. In: *Formal Methods in System Design* (1st May 2021). URL: https://hal.archives-ouvertes.fr/hal-03538097.

International peer-reviewed conferences

[10] 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. Chicago / Virtual, United States, 18th Oct. 2021. DOI: 10.1145/3428278. URL: https://hal.inria.fr/hal-03134294.

[11] D. Mazzucato and C. Urban. 'Reduced Products of Abstract Domains for Fairness Certification of Neural Networks'. In: 28th Static Analysis Symposium (SAS 2021). Chicago, United States, 17th Oct. 2021. URL: https://hal.inria.fr/hal-03348036.

[12] 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, 18th May 2021. URL: https://hal.archives-ouvertes.fr/hal-03538067.

[13] 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.archives-ouvertes.fr/hal-03538088.

National peer-reviewed Conferences

[14] S. Légaré, J. Krivine and J. Feret. 'Distinguishing Context Dependent Events in Quotients of Causal Stories'. In: JOBIM 2021 - Journées Ouvertes en Biologie, Informatique et Mathématiques. Virtuel, France, 2021, pp. 54–61. URL: https://hal.inria.fr/hal-03389052.

Scientific book chapters

[15] 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*. 2021, pp. 1–76. URL: https://hal.inria.fr/hal-03388100.

[16] V. Danos, H. E. Khalloufi and J. Prat. 'Global Order Routing on Exchange Networks'. In: *Financial Cryptography and Data Security. FC 2021 International Workshops*. Vol. 12676. Lecture Notes in Computer Science. Springer, 17th Sept. 2021, pp. 207–226. DOI: 10.1007/978-3-662-63958-0_19. URL: https://hal.archives-ouvertes.fr/hal-03455981.

Edition (books, proceedings, special issue of a journal)

[17] C. Lhoussaine and J. Feret. *Special issue "Formal Method for Biological Systems Modelling"*. Vol. 9. Section Computational Biology. MDPI, 2021. URL: https://hal.inria.fr/hal-03542053.

Reports & preprints

[18] C. Urban and A. Miné. *A Review of Formal Methods applied to Machine Learning*. 7th Apr. 2021. URL: https://hal.sorbonne-universite.fr/hal-03192255.

[19] 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*. 10th Nov. 2021. URL: https://hal.archives-ouvertes.fr/hal-03423441.

Other scientific publications

[20] S. Légaré, J. Krivine, R. Harmer and J. Feret. 'Modelling Systems Biology Wide and Deep'. In: CMSB 2021 - 19th International Conference on Computational Methods in Systems Biology. CMSB. Bordeaux, France, 22nd Sept. 2021. URL: https://hal.archives-ouvertes.fr/hal-03357485.

12.3 Cited publications

[21] 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.

[22] C. Urban, M. Christakis, V. Wüstholz and F. Zhang. 'Perfectly Parallel Fairness Certification of Neural Networks'. In: *Proceedings of the ACM on Programming Languages* 4.OOPSLA (13th Nov. 2020), pp. 1–30. DOI: 10.1145/3428253. URL: https://hal.inria.fr/hal-03091870.

[23] H. Illous, M. Lemerre and X. Rival. 'Interprocedural Shape Analysis Using Separation Logic-based Transformer Summaries'. In: SAS 2020 - 27th Static Analysis Symposium. Chicago / Virtual, United States, 18th Nov. 2020. URL: https://hal.inria.fr/hal-03081558.

[24] X. Rival and K. Yi. *Introduction to Static Analysis*. Feb. 2020. URL: https://hal.archives-ouvertes.fr/hal-02402597.

[25] J. Feret. *Analyses des motifs accessiblesdans les modèles Kappa*. 2020. URL: https://hal.inria.fr/hal-03088539.

[26] 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.

[27] 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.

[28] H. Illous. 'Abstract Heap Relations for a Compositional Shape Analysis'. Theses. Ecole Normale Supérieure, Apr. 2019. URL: https://hal.inria.fr/tel-02399767.

[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.