IN PARTNERSHIP WITH:
CNRS

Ecole normale supérieure de Paris

Activity Report 2019

Project-Team ANTIQUE

Static Analysis by Abstract Interpretation

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

RESEARCH CENTER
Paris

THEME
Proofs and Verification

Table of contents

Project-Team ANTIQUE

Creation of the Team: 2014 January 01, updated into Project-Team: 2015 April 01

Keywords:

Computer Science and Digital Science:

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, Visitors, External Collaborators

Research Scientists

Xavier Rival [Team leader, Inria, Senior Researcher, HDR]
Vincent Danos [CNRS, Senior Researcher, HDR]
Cezara Drăgoi [Inria, Researcher]

Jérôme Feret [Inria, Researcher]
Caterina Urban [Inria, Researcher, from Feb 2019]

PhD Students
Andreea Beica [Inria, PhD Student, until Mar 2019]
Gaëlle Candel [Keymetrics, PhD Student]
Marc Chevalier [Ecole Normale Supérieure Paris, PhD Student]
Patricio Inzaghi [Inria, PhD Student, from Jun 2019]
Olivier Nicole [CEA, PhD Student, from Sep 2019]
Albin Salazar [Inria, PhD Student, from Sep 2019]

Technical staff
Tie Cheng [Inria, Engineer, from May 2019]
Yves-Stan Le Cornec [Inria, Engineer, from Apr 2019]
Thierry Martinez [Inria, Engineer, from Mar 2019]

Interns and Apprentices
Jerome Boillot [Inria, from Jun 2019 until Jul 2019]
Guillaume Cluzel [Inria, until Jun 2019]
Lucas Escot [Ecole Normale Supérieure Lyon, until Jun 2019]
Dan Radulescu [Inria, from Jun 2019 until Sep 2019]
Vincent Rebiscoul [Inria, until Jun 2019]
Yvan Sraka [Inria, from Feb 2019 until Jul 2019]

Administrative Assistants
Chantal Chazelas [Inria, Administrative Assistant, until Jun 2019]
Nathalie Gaudechoux [Inria, Administrative Assistant]

Visiting Scientist
Pierre Boutillier [Harvard Medical School]

2. Overall Objectives

2.1. 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* [22]. 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* [22], 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 [21], 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. Models of growth

In systems and synthetic biology (engineered systems) one would like study the environment of a given cellular process (such as signaling pathways mentioned earlier) and the ways in which that process interacts with different resources provided by the host. To do this, we have built coarse-grained models of cellular physiology which summarize fundamental processes (transcription, translation, transport, metabolism). such models describe global growth in mechanistic way and allow one to plug the model of one's process of interest into a simplified and yet realistic and reactive model of the process interaction with its immediate environment. A first ODE-based deterministic version of this model [26] explaining the famous bacterial growth laws and how the allocation of resources to different genomic sectors depends on the growth conditions- was published in 2015 and has already received nearly 150 citations. The model also allows one to bridge between population genetic models which describe cells in terms of abstract features and fitness and intra-cellular models. For instance, we find that fastest growing strategies are not evolutionary stable in competitive experiments. We also find that vastly different energy storage strategies exist [24]. In a recent article [25] in *Nature Communications* we build a stochastic version of the above model. We predict the empirical size and doubling time distributions as a function of growth conditions. To be able to fit the parameters of the model to available single-cell data (note that the fitting constraints are far tighter than in the deterministic case), we introduce new techniques for the approximation of reaction-division systems which generalize continuous approximations of Langevin type commonly used for pure reaction systems. We also use cross-correlations to visualize causality and modes in noise propagation in the model (in a way reminiscent to abstract computational traces mentioned earlier). In other work, we show how to connect our new class of models to more traditional ones stemming from "flux balance analysis" by introducing an allocation vector which allows one to assign a formal growth rate to a class of reaction systems [20].

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 [13].

5. Highlights of the Year

5.1. Highlights of the Year

Caterina Urban joined the group as a CR in February 2019, and is opening new research directions towards static analysis for data-science software. She was invited to talk about her work in this area at SAS 2019 (Static Analysis Symposium) [13].

6. New Software and Platforms

6.1. 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.

- Participants: Antoine Miné and Bertrand Jeannet
- Contact: Antoine Miné
- URL: http://apron.cri.ensmp.fr/library/

6.2. Astrée

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.

- Participants: Antoine Miné, Jérôme Feret, Laurent Mauborgne, Patrick Cousot, Radhia Cousot and Xavier Rival
- Partners: CNRS - ENS Paris - AbsInt Angewandte Informatik GmbH
- Contact: Patrick Cousot
- URL: http://www.astree.ens.fr/

6.3. AstréeA

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.

- Participants: Antoine Miné, Jérôme Feret, Patrick Cousot, Radhia Cousot and Xavier Rival
- Partners: CNRS - ENS Paris - AbsInt Angewandte Informatik GmbH
- Contact: Patrick Cousot
- URL: http://www.astreea.ens.fr/

6.4. 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.

- Participants: Devin Mccoughlin, François Berenger and Pippijn Van Steenhoven
- Contact: Xavier Rival
- URL: https://github.com/Antique-team/clangml/tree/master/clang

6.5. 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.

- Participants: Antoine Miné and Caterina Urban
- Contact: Caterina Urban
- URL: http://www.di.ens.fr/~urban/FuncTion.html

6.6. HOO

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.

- Participant: Arlen Cox
- Contact: Arlen Cox

6.7. MemCAD

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.

- Participants: Antoine Toubhans, François Berenger, Huisong Li and Xavier Rival
- Contact: Xavier Rival
- URL: http://www.di.ens.fr/~rival/memcad.html

6.8. KAPPA

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 FUNCTIONAL DESCRIPTION: 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).

- Participants: Jean Krivine, Jérôme Feret, Kim-Quyen Ly, Pierre Boutillier, Russ Harmer, Vincent Danos and Walter Fontana
- Partners: ENS Lyon - Université Paris-Diderot - HARVARD Medical School
- Contact: Jérôme Feret
- URL: http://www.kappalanguage.org/

6.9. 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.

- Participant: Arlen Cox
- Contact: Arlen Cox

6.10. LCertify

KEYWORD: Compilation

SCIENTIFIC DESCRIPTION: The compilation certification process is performed automatically, thanks to a prover designed specifically. The automatic proof is done at a level of abstraction which has been defined so that the result of the proof of equivalence is strong enough for the goals mentioned above and so that the proof obligations can be solved by efficient algorithms.

FUNCTIONAL DESCRIPTION: Abstract interpretation, Certified compilation, Static analysis, Translation validation, Verifier. The main goal of this software project is to make it possible to certify automatically the compilation of large safety critical software, by proving that the compiled code is correct with respect to the source code: When the proof succeeds, this guarantees semantic equivalence. Furthermore, this approach should allow to meet some domain specific software qualification criteria (such as those in DO-178 regulations for avionics software), since it allows proving that successive development levels are correct with respect to each other i.e., that they implement the same specification. Last, this technique also justifies the use of source level static analyses, even when an assembly level certification would be required, since it establishes separately that the source and the compiled code are equivalent.ntees that no compiler bug did cause incorrect code to be generated.

- Participant: Xavier Rival
- Partners: CNRS - ENS Paris
- Contact: Xavier Rival
- URL: http://www.di.ens.fr/~rival/lcertify.html

6.11. 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.

- Participants: Antoine Miné, Pascal Cuoq and Xavier Leroy
- Contact: Antoine Miné
- URL: http://forge.ocamlcore.org/projects/zarith

6.12. PYPPAI

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.

- Contact: Xavier Rival
- URL: https://github.com/wonyeol/static-analysis-for-support-match

7. New Results

7.1. Relational Static Analysis

7.1.1. 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 done as part of the Phd of Hugo Illous [10] and a journal paper is currently under submission.

7.2. Static Analysis of Probabilistic Programming Languages

7.2.1. 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.

This work has been published in [12].

7.3. Static Analysis of JavaScript Code

7.3.1. Weakly Sensitive Analysis for Unbounded Iteration over JavaScript Objects

Participants: Yoonseok Ko, Xavier Rival [correspondant], Sukyoung Ryu.

In [23] and [11], we studied composite object abstraction for the analysis JavaScript.

JavaScript framework libraries like jQuery are widely use, but complicate program analyses. Indeed, they encode clean high-level constructions such as class inheritance via dynamic object copies and transformations that are harder to reason about. One common pattern used in them consists of loops that copy or transform part or all of the fields of an object. Such loops are challenging to analyze precisely, due to weak updates and as unrolling techniques do not always apply. In this work, we observe that precise field correspondence relations are required for client analyses (e.g., for call-graph construction), and propose abstractions of objects and program executions that allow to reason separately about the effect of distinct iterations without resorting to full unrolling. We formalize and implement an analysis based on this technique. We assess the performance and precision on the computation of call-graph information on examples from jQuery tutorials.

7.4. Rule-based Modeling with Arithmetics

7.4.1. Counters in Kappa: Semantics, Simulation, and Static Analysis.

Participants: Pierre Boutillier, Ioana Cristescu, Jérôme Feret.

Site-graph rewriting languages, such as Kappa or BNGL, offer parsimonious ways to describe highly combinatorial systems of mechanistic interactions among proteins. These systems may be then simulated efficiently. Yet, the modeling mechanisms that involve counting (a number of phosphorylated sites for instance) require an exponential number of rules in Kappa. In BNGL, updating the set of the potential applications of rules in the current state of the system comes down to the sub-graph isomorphism problem (which is NP-complete).

In [14], we extend Kappa to deal both parsimoniously and efficiently with counters. We propose a single push-out semantics for Kappa with counters. We show how to compile Kappa with counters into Kappa without counters (without requiring an exponential number of rules). We design a static analysis, based on affine relationships, to identify the meaning of counters and bound their ranges accordingly.

7.5. Reduced product

7.5.1. Sharing Ghost Variables in a Collection of Abstract Domains.

Participants: Marc Chevalier, Jérôme Feret.

In abstract interpretation, it is often necessary to be able to express complex properties while doing a precise analysis. A way to achieve that is to combine a collection of domains, each handling some kind of properties, using a reduced product. Separating domains allows an easier and more modular implementation, and eases soundness and termination proofs. This way, we can add a domain for any kind of property that is interesting. The reduced product, or an approximation of it, is in charge of refining abstract states, making the analysis precise.

In program verification, ghost variables can be used to ease proofs of properties by storing intermediate values that do not appear directly in the execution.

In [15], we propose a reduced product of abstract domains that allows domains to use ghost variables to ease the representation of their internal state. Domains must be totally agnostic with respect to other existing domains. In particular the handling of ghost variables must be entirely decentralized while still ensuring soundness and termination of the analysis.

7.6. Static Analysis of Neural Networks

7.6.1. 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 [19], 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.

7.7. Reductions between synchronous and asynchronous programming abstractions

7.7.1. Communication closed asynchronous protocols.

Participants: Andrei Damien, Cezara Drăgoi, Alexandru Militaru, Josef Widder.

Fault-tolerant distributed systems are implemented over asynchronous networks, where performance emerges from the load of the system. Due to asynchronous communication and the occurrence of faults (e.g., process crashes or the network dropping messages) the implementations are hard to understand and analyze. In contrast, synchronous computation models simplify design and reasoning.

In [17], we defined the first algorithm that automatically transforms an asynchronous protocol into a synchronous one. The method is sound but not complete. The transformation is based on an axiomatization of the notion of communication closure introduce by Elrad and Frances. If the asynchronous protocol is communication-closed then the translator will successfully compute its synchronous counter-part. Checking communication closure is done locally without considering any interferences between processes. The translator was successfully applied to Multi-Paxos, ViewStamped, and the atomic broadcast of Chandra and Toueg, generating the first synchronous counterparts of these protocols. The transformation from asynchronous to synchronous preserves the local states process go through and the exchanged messages. The translator has been implemented in a prototype tool called Athos, i.e., Asynchronous To Heard-Of Synchronizer, that is open source. The tool takes as input protocols in an intermediate protocol languages that has an asynchronous semantics and it is very close to C. These results have been published in one of the main verification venues Computer Aided Verification, CAV 2019 (acceptance rate <25% out of >250 submissions). The impact of the translator from asynchronous protocols to equivalent synchronous ones is important for the verification community because such a transformation reduces dramatically the state space and the set of traces to explore in order to prove the program correct, independently of the used verification technique.

7.7.2. Executable Rounds: a Programming Abstraction for Fault-Tolerant Protocols.

Participants: Cezara Drăgoi, Josef Widder, Damien Zufferey.

Fault-tolerant distributed systems are notoriously difficult to design and implement. Although programming languages for distributed systems is an active research area, appropriate synchronization primitives for fault-tolerance and group communication remains an important challenge. In [18] we present a new programming abstraction, HSync, for implementing benign and Byzantine distributed protocols. HSync is based on communication-closed rounds. Round models offer a simple abstraction for group communication and communication-closed rounds simplify dealing with faults. Protocols are implemented in a modular way in HSync. The language separates the message reception from the process local computation. It extends classic rounds with language constructs that give to the programmer the possibility to implement network and algorithm-specific policies for message reception. We have implemented an execution platform for HSync that runs on top of commodity hardware. We evaluate experimentally its performance, by comparing consensus implementations in HSync with LibPaxos3 and Bft-SMaRt, two consensus libraries tolerant to benign, resp. Byzantine faults.

7.8. Introduction

Participant: Andreea Beica.

The PhD of Andreea Beica [9] aims at studying two aspects related to the modelling of Biochemical Reaction Net- works, in the context of Systems Biology.

In the first part, we analyse how scale-separation in biological systems can be exploited for model reduction. We first argue for the use of rule-based models for prototyping genetic circuits, and then show how the inherent multi-scaleness of such systems can be used to devise a general model approximation method for rule-based models of genetic regulatory networks. The reduction proceeds via static analysis of the rule system. Our method relies on solid physical justifications, however not unlike other scale-separation reduction techniques, it lacks precise methods for quantifying the approximation error, while avoiding to solve the original model. Consequently, we next propose an approximation method for deterministic models of biochemical networks, in which reduction guarantees represent the major requirement. This second method combines abstraction and numerical approximation, and aims at providing a better understanding of model reduction methods that are based on time- and concentration- scale separation.

In the second part of the thesis, we introduce a new re-parametrisation technique for differential equation models of biochemical networks, in order to study the effect of intracellular resource storage strategies on growth, in self-replicating mechanistic models. Finally, we aim towards the characterisation of cellular growth as an emergent property of a novel Petri Net model semantics of Biochemical Reaction Networks.

8. Bilateral Contracts and Grants with Industry

8.1. Bilateral Contracts with Industry

8.1.1. Follow up to the AnaStaSec project

Title: Analyse de propriété de sécurité

Type: Research contracts funded by AirBus France

Duration: March 2019 - August 2018 and November 2019 - March 2020

Inria contact: Jérôme Feret

Abstract: An emerging structure in our information processing-based society is the notion of trusted complex systems interacting via heterogeneous networks with an open, mostly untrusted world. This view characterises a wide variety of systems ranging from the information system of a company to the connected components of a private house, all of which have to be connected with the outside.

The goal of these constracts is to analyse an application that is used to filter messages from higher-level security regions to lower-level ones in trusted complex systems. This application shall check that messages are well-formed and that they match with existing requests. Moreover, so as to limit potential flows of information, one shall prove that the internal state of buffers are reset between the processing of each packet.

To certify these properties, the front-end of ASTRÉE has been upgraded with new directives to specify the properties of interest, and the analysis has been tuned to improve the analysis : 1) ghost variables are used to record the value of buffers between each packet processing so that already existing relational domains can prove that they are restored to the correct value, and 2) data-partitioning strategies have been implemented to separate the different modes of usage.

9. Partnerships and Cooperations

9.1. National Initiatives

9.1.1. AnaStaSec

Title: Static Analysis for Security Properties

Type: ANR générique 2014

Defi: Société de l’information et de la communication

Instrument: ANR grant

Duration: January 2015 - September 2019

Coordinator: Inria Paris-Rocquencourt (France)

Others partners: Airbus France (France), AMOSSYS (France), CEA LIST (France), Inria Rennes-Bretagne Atlantique (France), TrustInSoft (France)

Inria contact: Jérôme Feret

See also: http://www.di.ens.fr/ feret/anastasec/

Abstract: An emerging structure in our information processing-based society is the notion of trusted complex systems interacting via heterogeneous networks with an open, mostly untrusted world. This view characterises a wide variety of systems ranging from the information system of a company to the connected components of a private house, all of which have to be connected with the outside.

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.

Some techniques have been developed and still need to be investigated to ensure security and confidentiality properties of such systems. Moreover, most of them are model-based techniques operating only at architectural level and provide no guarantee on the actual implementations. However, most security incidents are due to attackers exploiting subtle implementation-level software vulnerabilities. Systems should therefore be analyzed at software level as well (i.e. source or executable code), in order to provide formal assurance that security properties indeed hold for real systems.

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. The goal of this project is to develop the new concepts and technologies necessary to meet such a challenge.

The project AnaStaSec project will allow for the formal verification of security properties of software-intensive embedded systems, using automatic static analysis techniques at different levels of representation: models, source and binary codes. Among expected outcomes of the project will be a set of prototype tools, able to deal with realistic large systems and the elaboration of industrial security evaluation processes, based on static analysis.

9.1.2. DCore

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.

9.1.3. 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.

9.1.4. 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.

9.1.5. TGFSYSBIO

Title: Microenvironment and cancer: regulation of TGF-β signaling

Type: Plan Cancer 2014-2019

Duration: December 2015 - September 2019

Coordinator: INSERM U1085-IRSET

Others partners: Inria Paris (France), Inria Rennes-Bretagne Atlantique (France),

Inria contact: Jérôme Feret

Abstract: Most cases of hepatocellular carcinoma (HCC) develop in cirrhosis resulting from chronic liver diseases and the Transforming Growth Factor β (TGF-β) is widely regarded as both the major pro-fibrogenic agent and a critical inducer of tumor progression and invasion. Targeting the deleterious effects of TGF-β without affecting its physiological role is the common goal of therapeutic strategies. However, identification of specific targets remains challenging because of the pleiotropic effects of TGF-β linked to the complex nature of its extracellular activation and signaling networks.

Our project proposes a systemic approach aiming at to identifying the potential targets that regulate the shift from anti- to pro-oncogenic effects of TGF-β. To that purpose, we will combine a rule-based model (Kappa language) to describe extracellular TGF-beta activation and large-scale state-transition based (Cadbiom formalism) model for TGF-β-dependent intracellular signaling pathways. The multi-scale integrated model will be enriched with a large-scale analysis of liver tissues using shotgun proteomics to characterize protein networks from tumor microenvironment whose remodeling is responsible for extracellular activation of TGF-β. The trajectories and upstream regulators of the final model will be analyzed with symbolic model checking techniques and abstract interpretation combined with causality analysis. Candidates will be classified with semantic-based approaches and symbolic bi-clustering technics. All efforts must ultimately converge to experimental validations of hypotheses and we will use our hepatic cellular models (HCC cell lines and hepatic stellate cells) to screen inhibitors on the behaviors of TGF-β signal.

The expected results are the first model of extracellular and intracellular TGF-β system that might permit to analyze the behaviors of TGF-β activity during the course of liver tumor progression and to identify new biomarkers and potential therapeutic targets.

9.1.6. 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.

9.2. European Initiatives

9.2.1. FP7 & H2020 Projects

Type: IDEAS

Defi:

Instrument: ERC Proof of Concept Grant 2018

Objectif: Static Analysis for the VErification of Spreadsheets

Duration: January 2019 - June 2020

Coordinator: Inria (France)

Partner: None

Inria contact: Xavier Rival

Abstract: Spreadsheet applications (such as Microsoft Excel + VBA) are heavily used in a wide range of application domains including engineering, finance, management, statistics and health. However, they do not ensure robustness properties, thus spreadsheet errors are common and potentially costly. According to estimates, the annual cost of spreadsheet errors is around 7 billion dollars. For instance, in 2013, a series of spreadsheet errors at JPMorgan incurred 6 billion dollars trading losses. Yet, expert reports estimate about 90 % of the spreadsheets contain errors. The MemCAD ERC StG project opened the way to novel formal analysis techniques for spreadsheet applications. We propose to leverage these results into a toolbox able to safely *verify*, *optimize* and *maintain* spreadsheets, so as to reduce the likelihood of spreadsheet disasters. This toolbox will be commercialized by the startup MATRIXLEAD.

9.3. International Initiatives

9.3.1. Inria International Partners

9.3.1.1. Informal International Partners

Xavier Rival has a long standing collaboration with Bor-Yuh Evan Chang (University of Colorado, Boulder, USA), on the abstraction of symbolic properties and of complex memory data-structures.

Xavier Rival has a long standing collaboration with Sukyoung Ryu (KAIST, Daejeon, South Korea), on the analysis of dynamic programming languages. Xavier Rival has a set up a collaboration with Hongseok Yang (KAIST, Daejeon, South Korea), on the verification of probabilistic programs such as programs built in the Pyro framework.

Xavier Rival has started a collaboration with Shinya Katsumata, Jérémy Dubut, and Ichiro Hasuo (NII, Tokyo, Japan) on the formalization of abstract domains.

Xavier Rival has been working with Kwangkeun Yi on the writing of a book that should serve as an introduction to the field of static analysis, for students and engineers.

9.4. International Research Visitors

9.4.1. Visits of International Scientists

9.4.1.1. Internships

Marc Chevalier and Jérôme Feret have supervised the L3 internship of Jérôme Boillot (L3 at ENS Lyon).

Jérôme Feret has ksupervised the M2 internship of Yvan Sraka (M2 UPMC).

Xavier Rival has supervised M1 Internships of Guillaume Reboullet and of Luc Chabassier (M1 at DIENS).

Xavier Rival has supervised M2 Internships of Josselin Giet (MPRI at ENS) and of Vincent Rébiscoul (M2 at ENS Lyon).

9.4.2. Visits to International Teams

9.4.2.1. Research Stays Abroad

Xavier Rival has visited KAIST and Seoul National University in November 2019.

10. Dissemination

10.1. Promoting Scientific Activities

10.1.1. Scientific Events: Organisation

10.1.1.1. General Chair, Scientific Chair

- Jérôme Feret is a guest member of the Steering Committee of the Conference on Computational Methods in Systems Biology (CMSB).
- 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).

10.1.1.2. Member of the Organizing Committees

Caterina Urban serves as Chair of the Award Committee of the ETAPS Doctoral Dissertation Award 2020.

10.1.2. Event organization

- Cezara Drăgoi co-organize “VDS: Workshop on Verification of Distributed Systems” https://goto.ucsd.edu/vds/ with Klaus v. Gleissenthall from Univerisity of California, San Diego. This is a workshop that brings together two communities: verification and distributed systems by having participants from the two communities.

10.1.3. Scientific Events: Selection

10.1.3.1. Member of the Conference Program Committees

- Jérôme Feret served as a Member of the Program Committee of SASB 2019 (Static Analysis and Systems Biology).
- Jérôme Feret served as a Member of the Program Committee of CMSB 2019 (Computational Methods in Systems Biology).
- Jérôme Feret served as a Member of the Program Committee of CIBCB 2019 (Computational Intelligence in Bioinformatics and Computational Biology).
- Jérôme Feret served as a Member of the Program Committee of HSB 2019 (Hybrid Systems Biology).
- Jérôme Feret is serving as a member of the Program Committee of CMSB 2020 (Computational Methods in Systems Biology).

- Jérôme Feret is serving as a member of the Program Committee of HSB 2020 (Hybrid Systems Biology).
- Xavier Rival served as a Member of the Program Committee of SAS 2019 (Static Analysis Symposium).
- Xavier Rival is serving as a Member of the Program Committee of POPL 2020 (Symposium on Principles Of Programming Languages).
- Caterina Urban served as a member of the Program Committee of VMCAI 2019 (Verification, Model Checking, and Abstract Interpretation).
- Caterina Urban served as a member of the Artifact Evaluation Committee of POPL 2019 (Symposium on Principles of Programming Languages).
- Caterina Urban served as a member of the Program Committee of CAV 2019 (Computer Aided Verification).
- Caterina Urban served as a member of the Artifact Evaluation Committee of PLDI 2019 (Programming Languages Design and Implementation).
- Caterina Urban served as a member of the Program Committee of NSV 2019 (International Workshop on Numerical Software Verification).
- Caterina Urban served as a member of the Program Committee of LOPSTR 2019 (Logic-based Program Synthesis and Transformation).
- Caterina Urban served as a member of the Program Committee of TAPAS 2019 (Workshop on Tools for Automatic Program Analysis).
- Caterina Urban served as a member of the Program Committee of EMSOFT 2019 (Embedded Software).
- Caterina Urban served as a member of the Program Committee of iFM 2019 (integrated Formal Methods).
- Caterina Urban serves as a member of the Program Committee of VMCAI 2020 (Verification, Model Checking, and Abstract Interpretation).
- Caterina Urban serves as a member of the Program Committee of ESOP 2020 (European Symposium on Programming).
- Caterina Urban serves as a member of the Program Committee of CAV 2020 (Computer Aided Verification).
- Caterina Urban serves as a member of the Program Committee of SOAP 2020 (Workshop on the State Of the Art in Program Analysis).
- Cezara Drăgoi served as a member of the Program Committee of POPL 2020 (Symposium on Principles of Programming Languages)
- Cezara Drăgoi served as a member of the Program Committee of PLDI 2020 (Symposium on Programming Languages Design and Implementations)
- Cezara Drăgoi served as a member of the Program Committee of CAV 2019 (Computer Aided Verification).
- Cezara Drăgoi served as a member of the Program Committee of VMCAI 2019 (Verification, Model Checking, and Abstract Interpretation).
- Cezara Drăgoi served as a member of the Program Committee of NETYS 2019 (Conference on Networked Systems).

10.1.3.2. Reviewer

- Marc Chevalier served as Reviewer for SAS 2019 (Static Analysis Symposium).
- Marc Chevalier served as Reviewer for VMCAI 2020 (Verification, Model Checking, and Abstract Interpretation).

- Jérôme feret served as Reviewer for CAV 2019 (computer aided verification).
- Xavier Rival served as Reviewer for ESOP 2020 (European Symposium on Programming).
- Caterina Urban served as Reviewer for NFM 2019 (NASA Formal Methods).
- Caterina Urban served as Reviewer for POPL 2020 (Symposium on Principles of Programming Languages).

10.1.4. Journal

10.1.4.1. Member of the Editorial Boards

- Jérôme Feret serves as a Member of the Editorial Board of the Frontiers in Genetics journal and the Open Journal of Modeling and Simulation.
- Jérôme Feret served as co-Editor of an Issue of the Theoretical Computer Science journal, that is composed of papers from SASB 2016 and VEMP 2016, and appeared in 2019.
- Jérôme Feret served as co-Editor of an Issue of the IEEE/ACM Transactions on Computational Biology and Bioinformatics, that is composed of papers from CMSB 2017, and appeared in 2019.

10.1.4.2. Reviewer - Reviewing Activities

- Jérôme Feret served as a Reviewer for Fundamenta Informaticæ.
- Jérôme Feret served as a Reviewer for Theoretical Computer Science.
- Jérôme Feret served as a Reviewer for Transactions on Computational Biology and Bioinformatics.

10.1.5. Invited Talks

- Caterina Urban was invited to give a talk at the Gran Sasso Science Institute, L'Aquila, Italy.
- Caterina Urban was invited to give a talk at SAS 2019 (Static Analysis Symposium) [13].

10.1.6. Leadership within the Scientific Community

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

10.1.7. Scientific Expertise

- Jérôme Feret reviewed 5 proposals for the Prin 2017 (Research Projects of National Relevance) Program (Italian Ministry of Education and Research)

10.1.8. 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 deputy head of study of the Department of Computer Science of École normale supérieure.

10.2. Teaching - Supervision - Juries

10.2.1. Teaching

Licence:

- Marc Chevalier, Mathematics, 40h, L1, FDV Bachelor program (Frontiers in Life Sciences (FdV)), Université Paris-Descartes, France.
- Jérôme Feret and Xavier Rival (lectures), and Marc Chevalier (tutorials), "Semantics and Application to Verification", 36h, L3, at École Normale Supérieure, France.
- Xavier Rival, "Introduction to Static Analysis", 8h, L3, at École des Mines de Paris, France.
- Xavier Rival, "Functional Programming", 21h, L3, Bachelor Programme at at École Polytechnique, France.

Master:

- Xavier Rival, “Introduction to Static Analysis”, 12h, Internet of Things Master (retraining curriculum, EXED), France.
- Cezara Drăgoi, Jérôme Feret, Antoine Miné, and Xavier Rival, “Abstract Interpretation: application to verification and static analysis”, 72h, M2. Parisian Master of Research in Computer Science (MPRI), France.
- Vincent Danos and Jérôme Feret (with Jean Krivine), Rule-based Modelling, 24h, M1. Interdisciplinary Approaches to Life Science (AIV), Master Program, Université Paris-Descartes, France.

Doctorat:

- Jérôme Feret, “Abstract Interpretation of Models of Intracellular Signalling Pathways", 3h, CNRS summer school “Formal modeling of Biological Regulation Networks”, June 2019, Île de Porquerolles, France.

10.2.2. Supervision

- PhD defended: Andreea Beica, Abstraction of Biochemical Reaction Networks, started in 2015 and supervised by Vincent Danos
- PhD defended: Hugo Illous, Relational Shape Abstraction Based on Separation Logic, started in 2015 and supervised by Xavier Rival and Matthieu Lemerre (CEA)
- PhD in progress
- PhD in progress: Marc Chevalier, Static analysis of Security Properties in Critical Embedded Software, started in 2017 and supervised by 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: Olivier Nicole, Verification of micro-kernels, started in 2018 and supervised by Xavier Rival and Matthieu Lemerre (CEA)

10.2.3. Juries

- Jérôme Feret served as a member for the Jury of the PhD of Andreea Beica (Defense on the 12th of June 2019)
- Xavier Rival served as a member for the Habilitation Thesis Committee of Chien Chung Huang (Defense on the 28th of January 2019)
- Xavier Rival is serving as a member of the Review Committee for Vincenzo Arceri at University of Verona (to be defended in early 2020).
- Caterina Urban served as a member of the Jury of the PhD of Emilio Incerto at the Gran Sasso Science Institute in Italy (defense on the 5th of April 2019).
- Cezara Drăgoi servered as a member for the Jury of the PhD of Hugo Illous, Relational Shape Abstraction Based on Separation Logic (defence on the 16th avril 2019).

10.3. Popularization

10.3.1. Internal or external Inria responsibilities

- Cezara Drăgoi and Xavier Rival are elected members of the Inria Commision of Evaluation
- Jérôme Feret is a member of the jury of the SIF - Gilles Kahn PhD Award.
- Xavier Rival is member of the “Bureau du comité des projets” since October 2018.
- Xavier Rival is Chair of the Hiring Committee for Inria researchers at the center of Paris (CRCN) for 2019.

- Cezara Drăgoi served in the jury for part-time positions at LiX, January 2019.
- Cezara Drăgoi served in the Inria jury for CES - delegations and Phd scholarships in 2019.

10.3.2. Articles and contents

- Xavier Rival has been working with Kwangkeun Yi on the writing of a book that should serve as an introduction to the field of static analysis, for students and engineers, and this book is expected to be released by MIT Press in January 2020 [16].

10.3.3. Creation of media or tools for science outreach

Xavier Rival has given a talk as part of the “Mon équipe en 180 secondes” series at Inria Paris.

11. Bibliography

Major publications by the team in recent years

[1] J. BERTRANE, P. COUSOT, R. COUSOT, J. FERET, L. MAUBORGNE, A. MINÉ, 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, 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 7–14 2003, pp. 196–207

[3] A. BOUAJJANI, C. DRAGOI, C. ENEA, 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, 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", 2002, vol. 277, n^{o} 1–2, pp. 47–103

[5] J. FERET, V. DANOS, J. KRIVINE, R. HARMER, W. FONTANA. *Internal coarse-graining of molecular systems*, in "Proceeding of the national academy of sciences", Apr 2009, vol. 106, n^{o} 16

[6] L. MAUBORGNE, X. RIVAL. *Trace Partitioning in Abstract Interpretation Based Static Analyzers*, in "Proceedings of the 14th European Symposium on Programming (ESOP'05)", M. SAGIV (editor), Lecture Notes in Computer Science, Springer-Verlag, 2005, vol. 3444, pp. 5–20

[7] A. MINÉ. *The Octagon Abstract Domain*, in "Higher-Order and Symbolic Computation", 2006, vol. 19, 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

Publications of the year

Doctoral Dissertations and Habilitation Theses

[9] A. BEICA. *Abstractions of Biochemical Reaction Networks*, PSL University, June 2019, https://hal.inria.fr/tel-02411487

[10] H. ILLOUS. *Abstract Heap Relations for a Compositional Shape Analysis*, Ecole Normale Supérieure, April 2019, https://hal.inria.fr/tel-02399767

Articles in International Peer-Reviewed Journals

[11] Y. KO, X. RIVAL, S. RYU. *Weakly Sensitive Analysis for JavaScript Object-Manipulating Programs*, in "International Journal on Software - Practice and Experience", January 2019 [*DOI :* 10.1002/SPE], https://hal.archives-ouvertes.fr/hal-02399944

[12] W. LEE, H. YU, X. RIVAL, H. YANG. *Towards Verified Stochastic Variational Inference for Probabilistic Programs*, in "Proc. ACM Program. Lang", 2020, vol. 16, forthcoming [*DOI :* 10.1145/3371084], https://hal.archives-ouvertes.fr/hal-02399922

Invited Conferences

[13] C. URBAN. *Static Analysis of Data Science Software*, in "SAS 2019 - 26th Static Analysis Symposium", Porto, Portugal, B.-Y. E. CHANG (editor), Springer, October 2019, pp. 17-23 [*DOI :* 10.1007/978-3-030-32304-2_2], https://hal.inria.fr/hal-02397699

International Conferences with Proceedings

[14] P. BOUTILLIER, I. CRISTESCU, J. FERET. *Counters in Kappa: Semantics, Simulation, and Static Analysis*, in "ESOP 2019 - 28th European Symposium on Programming", Prague, Czech Republic, Springer, April 2019, pp. 176-204 [*DOI :* 10.1007/978-3-030-17184-1_7], https://hal.inria.fr/hal-02397876

[15] M. CHEVALIER, J. FERET. *Sharing Ghost Variables in a Collection of Abstract Domains*, in "VMCAI 2020 - 21st International Conference on Verification, Model Checking, and Abstract Interpretation", New Orleans, LA, United States, Proceedings of the 21st International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2020), January 2020, https://hal.inria.fr/hal-02378809

Scientific Books (or Scientific Book chapters)

[16] X. RIVAL, K. YI. *Introduction to Static Analysis*, MIT Press, February 2020, https://hal.archives-ouvertes.fr/hal-02402597

Other Publications

[17] A. DAMIAN, C. DRAGOI, A. MILITARU, J. WIDDER. *Communication-closed asynchronous protocols*, January 2019, working paper or preprint, https://hal.inria.fr/hal-01991415

[18] C. DRAGOI, J. WIDDER, D. ZUFFEREY. *Executable Rounds: a Programming Abstraction for Fault-Tolerant Protocols*, October 2019, working paper or preprint, https://hal.archives-ouvertes.fr/hal-02317446

[19] C. URBAN, M. CHRISTAKIS, V. WÜSTHOLZ, F. ZHANG. *Perfectly Parallel Fairness Certification of Neural Networks*, December 2019, working paper or preprint, https://hal.inria.fr/hal-02404036

References in notes

[20] A. BEICA, V. DANOS. *Synchronous Balanced Analysis*, in "Proceedings of the International Workshop on Hybrid Systems Biology", Springer-Verlag, Berlin, Germany, September 2016, pp. 85-94, https://hal.archives-ouvertes.fr/hal-01974696

[21] P. COUSOT. *Constructive design of a hierarchy of semantics of a transition system by abstract interpretation*, in "Electr. Notes Theor. Comput. Sci.", 1997, vol. 6, pp. 77–102, http://dx.doi.org/10.1016/S1571-0661(05)80168-9

[22] P. COUSOT, 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

[23] Y. KO, X. RIVAL, S. RYU. *Weakly Sensitive Analysis for Unbounded Iteration over JavaScript Objects*, in "APLAS 2017 - 15th Asian Symposium on Programming Languages and Systems", Suzhou, China, LNCS, Springer, November 2017, vol. 10695, pp. 148-168 [*DOI :* 10.1007/978-3-319-71237-6_8], https://hal.inria.fr/hal-01648680

[24] G. TERRADOT, A. BEICA, A. Y. WEISSE, V. DANOS. *Survival of the Fattest: Evolutionary Trade-offs in Cellular Resource Storage*, in "Electronic Notes in Theoretical Computer Science", April 2018, vol. 335, pp. 91-112 [*DOI :* 10.1016/J.ENTCS.2018.03.010], https://hal.archives-ouvertes.fr/hal-01976385

[25] P. THOMAS, G. TERRADOT, V. DANOS, A. Y. WEISSE. *Sources, propagation and consequences of stochasticity in cellular growth*, in "Nature Communications", December 2018, vol. 9, n^{o} 1 [*DOI :* 10.1038/S41467-018-06912-9], https://hal.archives-ouvertes.fr/hal-01976406

[26] A. Y. WEISSE, D. A. OYARZUN, V. DANOS, P. S. SWAIN. *Mechanistic links between cellular trade-offs, gene expression, and growth*, in "Proceedings of the National Academy of Sciences of the United States of America ", March 2015, vol. 112, n^{o} 9, pp. E1038-E1047 [*DOI :* 10.1073/PNAS.1416533112], https://hal.archives-ouvertes.fr/hal-01976394