IN PARTNERSHIP WITH:
CNRS

Ecole normale supérieure de Paris

Activity Report 2016

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:

2. - Software
2.1. - Programming Languages
2.1.1. - Semantics of programming languages
2.2.1. - Static analysis
2.3.1. - Embedded systems
2.4. - Verification, reliability, certification
2.4.1. - Analysis
2.4.2. - Model-checking
2.4.3. - Proofs
4.4. - Security of equipment and software
4.5. - Formal methods for security

Other Research Topics and Application Domains:

1.1. - Biology
1.1.10. - Mathematical biology
1.1.11. - Systems biology
5.2. - Design and manufacturing
5.2.1. - Road vehicles
5.2.2. - Railway
5.2.3. - Aviation
5.2.4. - Aerospace
6.1. - Software industry
6.1.1. - Software engineering
6.1.2. - Software evolution, maintenance
6.6. - Embedded systems

1. Members

Research Scientists

Xavier Rival [Team leader, Inria, Senior Researcher, HDR]
Patrick Cousot [ENS Paris, Professor Emeritus, HDR]
Vincent Danos [CNRS, Senior Researcher, HDR]
Cezara Dragoi [Inria, Researcher]
Jérôme Feret [Inria, Researcher]

Engineers

Francois Berenger [Inria]
Tie Cheng [Inria, until May 2016]
Kim Quyen Ly [Inria]

PhD Students

Mehdi Bouaziz [Inria, until Nov 2016]
Ferdinanda Camporesi [Inria]
Huisong Li [Inria]
Jiangchao Liu [Inria]
Hugo Illous [CEA, ENS]
Thibault Suzanne [ENS Paris]
Andreea Beica [ENS Paris]
Patric Fulop [University of Edinburgh and ENS Paris]

Post-Doctoral Fellows

Ilias Garnier [ENS Paris and University of Edinburgh]
Nicolas Behr [ENS Paris and University of Edinburgh]
Ricardo Honorato-Zimmer [ENS Paris and University of Edinburgh]

Visiting Scientist

Kwangkeun Yi [Seoul National University, until Oct 2016]

Administrative Assistant

Nathalie Gaudechoux [Inria]

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 became 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), and medical systems (pacemakers, surgery and patient monitoring systems) rely on complex software, which should be verified.

Beyond the field of embedded systems, other pieces of software may cause very significant harm in 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 be 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 expressing 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* [13]. 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* [13], which over-approximates 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.

Yet, the same principles can also be applied successfully to other goals. In particular, the abstract interpretation framework should be viewed 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 [12], 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. The analysis of biological models

One of our application domains, the analysis of biological models, is not a classical target of static analysis because it aims at analyzing models instead of programs. Yet, the analysis of biological models is closely intertwined with the other application fields of our group. Firstly, abstract interpretation provides a formal understanding of the abstraction process which is inherent to the modeling process. Abstract interpretation is also used to better understand the systematic approaches which are used in the systems biology field to capture the properties of models, until getting formal, fully automatic, and scalable methods. Secondly, abstract interpretation is used to offer various semantics with different grains of abstraction, and, thus, new methods to apprehend the overall behavior of the models. Conversely, some of the methods and abstractions which are developed for biological models are inspired by the analysis of concurrent systems and by security analysis. Lastly, the analysis of biological models raises issues about differential systems, stochastic systems, and hybrid systems. Any breakthrough in these directions will likely be very important to address the important challenge of the certification of critical systems in interaction with their physical environment.

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 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, monitoring typically rely on a *parallel* structure, where several threads are executed in parallel, 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. Biological systems

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 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 abstract interpretation-based analyses, 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 so as to identify the key elements in the dynamics of these models. The results of these analyses are sent to another tool, which is used to automatically simplify the 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.

5. Highlights of the Year

5.1. Highlights of the Year

The team obtained several strong results published in excellent international conferences, with high theoretical and applied impact(see detailed results). Among the theoretical results we underline those presented in conferences like Principles of programming languages POPL 2016, and among the applied results we underline the release of MemCad, the first analyzer that can handle the analysis of various data structures.

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

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 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: Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné and Xavier Rival
- Partner: CNRS
- Contact: Patrick Cousot
- URL: http://www.astree.ens.fr/

6.3. AstréeA

The AstréeA Static Analyzer of Asynchronous Software
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: Patrick Cousot, Radhia Cousot, Jérôme Feret, Antoine Miné and Xavier Rival est toujours membre de Inria. logiciels Inria): https://bil.inria.fr/
- Contact: Patrick Cousot
- URL: http://www.astreea.ens.fr/

6.4. ClangML

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: François Berenger, Pippijn Van Steenhoven and Devin Mccoughlin toujours membre de Inria. Inria): https://bil.inria.fr/
- Contact: François Berenger
- 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: Caterina Urban and Antoine Miné
- Contact: Caterina Urban
- URL: http://www.di.ens.fr/~urban/FuncTion.html

6.6. MemCAD

The MemCAD static analyzer
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 350 small size test cases that are used as regression tests.

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

6.7. OPENKAPPA

La platte-forme de modélisation OpenKappa
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.

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

6.8. 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

6.9. Translation Validation

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 guara 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
- Contact: Xavier Rival

6.10. Zarith

FUNCTIONAL DESCRIPTION

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

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

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

6.11. CELIA

The MemCAD static analyzer

FUNCTIONAL DESCRIPTION

CELIA is a tool for the static analysis and verification of C programs manipulating dynamic lists. The static analyzer computes for each control point of a C program the assertions which are true (i.e., invariant) at this control point. The specification language is a combination of Separation Logic with a first order logic over sequences of integers. The inferred properties describe the shape of the lists, their size, the relations between the data (or the sum, or the multiset of data) in list cells. The analysis is inter-procedural, i.e., the assertions computed relate the procedure local heap on entry to the corresponding local heap on exit of the procedure. The results of the analysis can provide insights about equivalence of procedures on lists or null pointer dereferencing. The analysis is currently extended to programs manipulating concurrent data structures.

- Participants: Ahmed Bouajjani, Cezara Drăgoi, Constantin Enea, Mihaela Sighireanu
- Contact: Cezara Drăgoi
- URL: http://www.liafa.jussieu.fr/celia/

6.12. DAFT

DAFT
FUNCTIONAL DESCRIPTION

DAFT is a distributed file management system in user-space, with a command-line interface. DAFT is intended at computational scientists, involved in data-intensive, distributed experiments and when no distributed file-system is available on computing nodes. DAFT is secure; all messages are cryptographically signed and encrypted by default.

- Participants: Francois Berenger and Camille Coti.
- Contact: Francois Berenger and Camille Coti.
- URL: https://github.com/UnixJunkie/daft.

7. New Results

7.1. Memory Abstraction

7.1.1. Abstraction of arrays based on non contiguous partitions

Participants: Jiangchao Liu, Xavier Rival [correspondant].

Abstract interpretation, Memory abstraction, Array abstract domains. In [2], we studied array abstractions.

Array partitioning analyses split arrays into contiguous partitions to infer properties of cell sets. Such analyses cannot group together non contiguous cells, even when they have similar properties. We proposed an abstract domain which utilizes semantic properties to split array cells into groups. Cells with similar properties will be packed into groups and abstracted together. Additionally, groups are not necessarily contiguous. This abstract domain allows to infer complex array invariants in a fully automatic way. Experiments on examples from the Minix 1.1 memory management demonstrated its effectiveness.

7.2. Rule-based modeling

7.2.1. Reachability analysis via orthogonal sets of patterns

Participants: Kim Quyên Ly, Jérôme Feret [correspondant].

Rule-based modeling languages, as Kappa, allow for the description of very detailed mechanistic models. Yet, as the rules become more and more numerous, there is a need for formal methods to enhance the level of confidence in the models that are described with these languages. We develop abstract interpretation tools to capture invariants about the biochemical structure of bio-molecular species that may occur in a given model. In previous works, we have focused on the relationships between the states of the sites that belong to a same instance of a protein. This comes down to detect for a specific set of patterns, which ones may be reachable during the execution of the model. This paper [6], we generalize this approach to a broader family of abstract domains, that we call orthogonal sets of patterns. More precisely, an orthogonal set of patterns is obtained by refining recursively the information about some patterns containing a given protein, so as to partition of the set of occurrences of this protein in any mixture.

7.2.2. Local traces: an over-approximation of the behaviour of the proteins in rule-based models

Participants: Kim Quyên Ly, Jérôme Feret [correspondant].

Thanks to rule-based modelling languages, we can assemble large sets of mechanistic protein-protein interactions within integrated models. Our goal would be to understand how the behaviour of these systems emerges from these low-level interactions. Yet this is a quite long term challenge and it is desirable to offer intermediary levels of abstraction, so as to get a better understanding of the models and to increase our confidence within our mechanistic assumptions. In this paper [5], we propose an abstract interpretation of the behaviour of each protein, in isolation. Given a model written in Kappa, this abstraction computes for each kind of protein a transition system that describes which conformations this protein can take and how a protein can pass from one conformation to another one. Then, we use simplicial complexes to abstract away the interleaving order of the transformations between conformations that commute. As a result, we get a compact summary of the potential behaviour of each protein of the model.

7.3. Formal Derivation of Qualitative Dynamical Models from Biochemical Networks

Participants: Wassim Abou-Jaoudé, Denis Thieffry, Jérôme Feret [correspondant].

As technological advances allow a better identification of cellular networks, more and more molecular data are produced allowing the construction of detailed molecular interaction maps. One strategy to get insights into the dynamical properties of such systems is to derive compact dynamical models from these maps, in order to ease the analysis of their dynamics. Starting from a case study, we present in [1] a methodology for the derivation of qualitative dynamical models from biochemical networks. Properties are formalised using abstract interpretation. We first abstract states and traces by quotienting the number of instances of chemical species by intervals. Since this abstraction is too coarse to reproduce the properties of interest, we refine it by introducing additional constraints. The resulting abstraction is able to identify the dynamical properties of interest in our case study.

7.4. Taking Static Analysis to the Next Level: Proving the Absence of Run-Time Errors and Data Races with Astrée

Participants: Antoine Miné, Laurent Mauborgne, Xavier Rival, Jérôme Feret [correspondant], Patrick Cousot, Daniel Kästner, Stephan Wilhelm, Christian Ferdinand.

In [9], we present an extension of Astrée to concurrent C software. Astrée is a sound static analyzer for run-time errors previously limited to sequential C software. Our extension employs a scalable abstraction which covers all possible thread interleavings, and soundly reports all run-time errors and data races: when the analyzer does not report any alarm, the program is proven free from those classes of errors. We show how this extension is able to support a variety of operating systems (such as POSIX threads, ARINC 653, OSEK/AUTOSAR) and report on experimental results obtained on concurrent software from different domains, including large industrial software.

7.5. Stochastic mechanics of graph rewriting

Participants: Nicolas Behr, Vincent Danos, Ilias Garnier [correspondant].

We propose an algebraic approach to stochastic graph-rewriting which extends the classical construction of the Heisenberg-Weyl algebra and its canonical representation on the Fock space. Rules are seen as particular elements of an algebra of “diagrams": the diagram algebra D. Diagrams can be thought of as formal computational traces represented in partial time. They can be evaluated to normal diagrams (each corresponding to a rule) and generate an associative unital non-commutative algebra of rules: the rule algebra R. Evaluation becomes a morphism of unital associative algebras which maps general diagrams in D to normal ones in R. In this algebraic reformulation, usual distinctions between graph observables (real-valued maps on the set of graphs defined by counting subgraphs) and rules disappear. Instead, natural algebraic substructures of R arise: formal observables are seen as rules with equal left and right hand sides and form a commutative subalgebra, the ones counting subgraphs forming a sub-subalgebra of identity rules. Actual graph-rewriting is recovered as a canonical representation of the rule algebra as linear operators over the vector space generated by (isomorphism classes of) finite graphs. The construction of the representation is in close analogy with and subsumes the classical (multi-type bosonic) Fock space representation of the Heisenberg-Weyl algebra.

This shift of point of view, away from its canonical representation to the rule algebra itself, has unexpected consequences. We find that natural variants of the evaluation morphism map give rise to concepts of graph transformations hitherto not considered. These will be described in a separate paper [2]. In this extended abstract we limit ourselves to the simplest concept of double-pushout rewriting (DPO). We establish "jump-closure", i.e. that the sub-space of representations of formal graph observables is closed under the action of any rule set. It follows that for any rule set, one can derive a formal and self-consistent Kolmogorov backward equation for (representations of) formal observables.

This result and the fallowing ones, co-authored by Vincent Danos, were published in peer-reviewed international conferences and journals. Although the papers are on HAL, they are not imported in the bibtex file so we can't cite them properly.

7.6. Giry and the machine

Participants: Fredrik Dahlqvist, Vincent Danos, Ilias Garnier [correspondant].

We present a general method – the Machine – to analyse and characterise in finitary terms natural transformations between well-known functors in the category Pol of Polish spaces. The method relies on a detailed analysis of the structure of Pol and a small set of categorical conditions on the domain and codomain functors. We apply the Machine to transformations from the Giry and positive measures functors to combinations of the Vietoris, multiset, Giry and positive measures functors. The multiset functor is shown to be defined in Pol and its properties established. We also show that for some combinations of these functors, there cannot exist more than one natural transformation between the functors, in particular the Giry monad has no natural transformations to itself apart from the identity. Finally we show how the Dirichlet and Poisson processes can be constructed with the Machine.

7.7. Robustly Parameterised Higher-Order Probabilistic Models

Participants: Fredrik Dahlqvist, Vincent Danos, Ilias Garnier [correspondant].

We present a method for constructing robustly parameterised families of higher-order probabilistic models. Parameter spaces and models are represented by certain classes of functors in the category of Polish spaces. Maps from parameter spaces to models (parameterisations) are continuous and natural transformations between such functors. Naturality ensures that parameterised models are invariant by change of granularity – i.e. that parameterisations are intrinsic. Continuity ensures that models are robust with respect to their parameterisation. Our method allows one to build models from a set of basic functors among which the Giry probabilistic functor, spaces of cadlag trajectories (in continuous and discrete time), multisets and compact powersets. These functors can be combined by guarded composition, product and coproduct. Parameter spaces range over the polynomial closure of Giry-like functors. Thus we obtain a class of robust parameterised models which includes the Dirichlet process, various point processes (random sequences with values in Polish spaces) and other classical objects of probability theory. By extending techniques developed in prior work, we show how to reduce the questions of existence, uniqueness, naturality, and continuity of a parameterised model to combinatorial questions only involving finite spaces.

7.8. Bayesian inversion by ω-complete cone duality

Participants: Fredrik Dahlqvist, Vincent Danos, Ilias Garnier [correspondant], Ohad Kammar.

The process of inverting Markov kernels relates to the important subject of Bayesian modelling and learning. In fact, Bayesian update is exactly kernel inversion. In this paper, we investigate how and when Markov kernels (aka stochastic relations, or probabilistic mappings, or simply kernels) can be inverted. We address the question both directly on the category of measurable spaces, and indirectly by interpreting kernels as Markov operators: For the direct option, we introduce a typed version of the category of Markov kernels and use the so-called 'disintegration of measures'. Here, one has to specialise to measurable spaces borne from a simple class of topological spaces -e.g. Polish spaces (other choices are possible). Our method and result greatly simplify a recent development in Ref. [4]. For the operator option, we use a cone version of the category of Markov

operators (kernels seen as predicate transformers). That is to say, our linear operators are not just continuous, but are required to satisfy the stronger condition of being ω-chain-continuous.1 Prior work shows that one obtains an adjunction in the form of a pair of contravariant and inverse functors between the categories of L1- and L^∞-cones [3]. Inversion, seen through the operator prism, is just adjunction.2 No topological assumption is needed. We show that both categories (Markov kernels and ω-chain-continuous Markov operators) are related by a family of contravariant functors Tp for $1 \leq p \leq \infty$. The Tp's are Kleisli extensions of (duals of) conditional expectation functors introduced in Ref. [3]. With this bridge in place, we can prove that both notions of inversion agree when both defined: if f is a kernel, and f† its direct inverse, then T∞(f)† = T1(f†).

7.9. Continuous-time Markov chains as transformers of unbounded observables

Participants: Vincent Danos, Ilias Garnier [correspondant], Tobias Heindel, Jakob Simonsen.

We provide broad sufficient conditions for the com-putability of time-dependent averages of stochastic processes of the form f (Xt) where Xt is a continuous-time Markov chain (CTMC), and f is a real-valued function (aka an observable). We consider chains with values in a countable state space S, and possibly unbounded f s. Observables are seen as generalised predicates on S and chains are interpreted as transformers of such generalised predicates, mapping each observable f to a new observable Ptf defined as (Ptf)(x) = Ex(f (Xt)), which represents the mean value of f at time time t as a function of the initial state x. We obtain three results. First, the well-definedness of this operator interpretation is obtained for a large class of chains and observables by restricting Pt to judiciously chosen rescalings of the basic Banach space C0(S) of S-indexed sequences which vanish at infinity. We prove, under appropriate assumptions, that the restricted family Pt forms a strongly continuous operator semigroup (equivalently the time evolution map t $\rightarrow$ Pt is continuous w.r.t. the usual topology on bounded operators). The computability of the time evolution map follows by generic arguments of constructive analysis. A key point here is that the assumptions are flexible enough to accommodate unbounded observables, and we give explicit examples of such using stochastic Petri nets and stochastic string rewriting. Thirdly, we show that if the rate matrix (aka the q-matrix) of the CTMC is locally algebraic on a subspace containing f , the time evolution of projections t $\rightarrow$ (Ptf)(x) is PTIME computable for each x. These results provide a functional analytic alternative to Monte Carlo simulation as test bed for mean-field approximations, moment closure, and similar techniques that are fast, but lack absolute error guarantees.

7.10. Communities in socio-cognitive networks.

Participants: Vincent Danos, Ricardo Honorato-Zimmer [correspondant].

We investigate a recent network model which combines social and cognitive features. Each node in the social network holds a (possibly different) cognitive network that represent its beliefs. In this internal cognitive network a node denotes a concept and a link indicates whether the two linked concepts are taken to be of a similar or opposite nature. We show how these networks naturally organise into communities and use this to develop a method that detects communities in social networks. How they organise depends on the social structure and the ratio between the cognitive and social forces driving the propagation of beliefs.

7.11. Synchronous Balanced Analysis

Participants: Andreea Beica [correspondant], Vincent Danos.

When modeling Chemical Reaction Networks, a commonly used mathematical formalism is that of Petri Nets, with the usual interleaving execution semantics. We aim to substitute to a Chemical Reaction Network, especially a "growth" one (i.e., for which an exponential stationary phase exists), a piecewise synchronous approximation of the dynamics: a resource-allocation-centered Petri Net with maximal-step execution semantics. In the case of unimolecular chemical reactions, we prove the correctness of our method and show that it can be used either as an approximation of the dynamics, or as a method of constraining the reaction rate constants (an alternative to flux balance analysis, using an emergent formally defined notion of "growth rate" as the objective function), or a technique of refuting models.

7.12. Pointless learning

Participants: Florence Clerc, Fredrik Dahlqvist, Vincent Danos, Ilias Garnier [correspondant].

Bayesian inversion is at the heart of probabilistic programming and more generally machine learning. Understanding inversion is made difficult by the pointful (kernel-centric) point of view usually taken in the literature. We develop in a pointless (kernel-free) approach to inver- sion. While doing so, we revisit some foundational objects of probability theory, unravel their category-theoretical underpinnings and show how pointless Bayesian inversion sits naturally at the centre of this construction.

7.13. Survival of the fattest.

Participants: Andreea Beica [correspondant], Vincent Danos, Guillaume Terradot, Andrea Weisse.

Cells derive resources from their environments and use them to fuel the biosynthetic processes that determine cell growth. Depending on how responsive the biosynthetic processes are to the availability of intracellular resources, cells can build up different levels of resource storage. Here we use a recent mathematical model of the coarse-grained mechanisms that drive cellular growth to investigate the effects of cellular resource storage on growth. We show that, on the one hand, there is a cost associated with high levels of storage resulting from the loss of stored resources due to dilution. We further show that, on the other hand, high levels of storage can benefit cells in variable environments by increasing biomass production during transitions from one medium to another. Our results thus suggest that cells may face trade-offs in their maintenance of resource storage based on the frequency of environmental change.

7.14. The algebras of graph rewriting

Participants: Nicolas Behr, Vincent Danos, Ilias Garnier [correspondant], Tobias Heindel.

The concept of diagrammatic combinatorial Hopf algebras in the form introduced for describing the Heisenberg-Weyl algebra is extended to the case of so-called rule diagrams that present graph rewriting rules and their composites. The resulting rule diagram algebra may then be suitably restricted in four different ways to what we call the rule algebras, which are non-commutative, unital associative algebras that implement the algebra of compositions of graph rewriting rules. Notably, our framework reveals that there exist two more types of graph rewriting systems than previously known in the literature, and we present an analysis of the structure of the rule algebras as well as a form of Poincaré-Birkhoff-Witt theorem for the rule diagram algebra. Our work lays the foundation for a fundamentally new way of analyzing graph transformation systems, and embeds this very important concept from theoretical computer science firmly into the realm of mathematical combinatorics and statistical physics.

7.15. PSYNC: A partially synchronous language for fault-tolerant distributed algorithms

Participants: Cezara Drăgoi [correspondant], Thomas Henzinger [IST Austria, Austria], Damien Zufferey [MIT, CSAIL, USA].

Fault-tolerant distributed systems, Programming languages, Verification Fault-tolerant distributed algorithms play an important role in many critical/high-availability applications. These algorithms are notori- ously difficult to implement correctly, due to asynchronous com- munication and the occurrence of faults, such as the network drop- ping messages or computers crashing. We introduce PSYNC in [4], a domain specific language based on the Heard-Of model, which views asynchronous faulty systems as syn- chronous ones with an adversarial environment that simulates asyn- chrony and faults by dropping messages. We define a runtime sys- tem for PSYNC that efficiently executes on asynchronous networks. We formalize the relation between the runtime system and PSYNC in terms of observational refinement. The high-level lockstep ab- straction introduced by PSYNC simplifies the design and imple- mentation of fault-tolerant distributed algorithms and enables auto- mated formal verification. We have implemented an embedding of PSYNC in the SCALA programming language with a runtime system for asynchronous networks. We show the applicability

of PSYNC by implementing several important fault-tolerant distributed algorithms and we com- pare the implementation of consensus algorithms in PSYNC against implementations in other languages in terms of code size, runtime efficiency, and verification.

8. Partnerships and Cooperations

8.1. National Initiatives

8.1.1. 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 - December 2018

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.

8.1.2. REPAS

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

8.1.3. VerAsCo

Title: Formally-verified static analyzers and compilers
Type: ANR Ingénierie Numérique Sécurité 2011
Instrument: ANR grant
Duration: September 2011 - June 2016
Coordinator: Inria (France)
Others partners: Airbus France (France), IRISA (France), Inria Saclay (France)
See also: http://www.systematic-paris-region.org/fr/projets/verasco
Abstract: The usefulness of verification tools in the development and certification of critical software is limited by the amount of trust one can have in their results. A first potential issue is *unsoundness* of a verification tool: if a verification tool fails (by mistake or by design) to account for all possible executions of the program under verification, it can conclude that the program is correct while it actually misbehaves when executed. A second, more insidious, issue is *miscompilation*: verification tools generally operate at the level of source code or executable model; a bug in the compilers and code generators that produce the executable code that actually runs can lead to a wrong executable being generated from a correct program.

The project VERASCOadvocates a mathematically-grounded solution to the issues of formal verifying compilers and verification tools. We set out to develop a generic static analyzer based on abstract interpretation for the C language, along with a number of advanced abstract domains and domain combination operators, and prove the soundness of this analyzer using the Coq proof assistant. Likewise, we will continue our work on the CompCert C formally-verified compiler, the first realistic C compiler that has been mechanically proved to be free of any miscompilation will be continued. Finally, the tool qualification issues that must be addressed before formally-verified tools can be used in the aircraft industry, will be investigated.

8.1.4. AstréeA

Title: Static Analysis of Embedded Asynchronous Real-Time Software
Type: ANR Ingénierie Numérique Sécurité 2011
Instrument: ANR grant
Duration: January 2012 - November 2016
Coordinator: Airbus France (France)
Others partners: École normale supérieure (France)
Inria contact: Antoine Miné
See also: http://www.astreea.ens.fr
Abstract: The focus of the ASTRÉEA project is on the development of static analysis by abstract interpretation to check the safety of large-scale asynchronous embedded software. During the THESEE ANR project (2006–2010), we developed a concrete and abstract models of the ARINC 653 operating system and its scheduler, and a first analyzer prototype. The gist of the ASTRÉEA project is the continuation of this effort, following the recipe that made the success of ASTRÉE: an incremental refinement of the analyzer until reaching the zero false alarm goal. The refinement concerns: the abstraction of process interactions (relational and history-sensitive abstractions), the scheduler model (supporting more synchronisation primitives and taking priorities into account), the memory model (supporting volatile variables), and the abstraction of dynamical data-structures (linked lists). Patrick Cousot is the principal investigator for this project.

8.1.5. VeriFault

This was a PEPS project for one year, coordinated by Cezara Drăgoi, on the topic of fault-tolerant distributed algorithms. These algorithms are notoriously difficult to implement correctly, due to asynchronous communication and the occurrence of faults, such as the network dropping messages or computers crashing. Although fault-tolerant algorithms are at the core of critical applications, there are no automated verification techniques that can deal with their complexity. Due to the complexity distributed systems have reached, we believe it is no longer realistic nor efficient to assume that high level specifications can be proved when development and verification are two disconnected steps in the software production process. Therefore we propose to introduce a domain specific language that has a high-level control structure which focuses on the algorithmic aspects rather than on low-level network and timer code, and makes programs amendable to automated verification.

8.2. European Initiatives

8.2.1. FP7 & H2020 Projects

ASSUME, ITEA 3 project (Affordable Safe & Secure Mobility Evolution). Affordable Safe & Secure Mobility Evolution

Future mobility solutions will increasingly rely on smart components that continuously monitor the environment and assume more and more responsibility for a convenient, safe and reliable operation. Currently the single most important roadblock for this market is the ability to come up with an affordable, safe multi-core development methodology that allows industry to deliver trustworthy new functions at competitive prices. ASSUME will provide a seamless engineering methodology, which addresses this roadblock on the constructive and analytic side.

8.3. International Research Visitors

8.3.1. Visits of International Scientists

Prof. Kwangkeun Yi Visiteur from Seoul National University, was an invited visitor until Oct 2016.

8.3.1.1. Internships

- Ken Chanseau Saint-Germain, ENS Paris, until Aug 2016
- Marc Chevalier, ENS Lyon, since Sept 2016
- Anton Kulaga, Jul and Aug 2016
- Yoon Seok Ko, Inria, until Jun 2016
- David Romero Suarez, Inria, from Feb 2016 until May 2016]
- Gaelle Candel, Chimie ParisTech

9. Dissemination

9.1. Promoting Scientific Activities

9.1.1. Scientific Events Organisation

9.1.1.1. General Chair, Scientific Chair

Jérôme Feret is a member of the editorial board of the Frontiers in Genetics journal and the Open Journal of Modeling and Simulation.

9.1.1.2. Member of the Organizing Committees

Jérôme Feret organized the 40th of Abstract Interpretation at POPL2017 January 21, 2017, Paris, France (co-organizer).

9.1.2. Scientific Events Selection

9.1.2.1. Chair of Conference Program Committees

Xavier Rival was chair of Static Analysis Symposium SAS 2016, Edinburgh.

Jérôme Feret co-chaired the fifteenth Conference on Computational Methods in Systems Biology - CMSB 2017, September 27–29, 2017, Darmstadt, Germany.

9.1.2.2. Member of the Conference Program Committees

Vincent Danos served on the PC of Computational Methods in Systems Biology, CMSB'16, Cambridge and Complexis'17.

Xavier Rival served on the PC of the 26th European Symposium on Programming (ESOP 2017).

Cezara Drăgoi served on the PC of

- 28th International Conference on Computer-Aided Verification (ERC), CAV 2016,
- 37th INTERNATIONAL CONFERENCE ON APPLICATIONS AND THEORY OF PETRI NETS AND CONCURRENCY, ACSD 2016,
- 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2017,
- 18th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2017,
- ACM SIGPLAN Symposium on Programming Language Design & Implementation, PLDI 2017.

Jérôme Feret served on the PC of

- the 8th International Conference on Bioinformatics, Biocomputational Systems and Biotechnologies - BIOTECHNO 2016,
- the 26th International Symposium on Logic-Based Program Synthesis and transformation - LOPSTR 2016,
- the 23rd Static Analysis Symposium Sept 8-10 2016, Edinburgh,
- 7th International Workshop on Static Analysis and Systems Biology - SASB 2016,
- 14th International Conference on Computational Methods in Systems Biology - CMSB 2016 Sept 21-23 2016, Cambridge, UK,
- Fourth International Conference on Tools and Methods for Program Analysis - TMPA 2017 March 3–4, 2017, Moscow, Russia, JOBIM 2017 July 2-6 2017, Lille, France.

9.1.2.3. Reviewer

Vincent Danos was a reviewer for the 19th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS) 2017, the 26th European Symposium on Programming (ESOP 2017) 2017, LMCS, MSCS.

Cezara Drăgoi was a reviewer for 19th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS) 2017, the 27th International Conference on Concurrency Theory CONCUR 2016, and the 26th European Symposium on Programming (ESOP 2017) 2017.

Xavier Rival was a reviewer for 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2017 and ACM SIGPLAN Symposium on Programming Language Design & Implementation, PLDI 2017.

Jérôme Feret was a reviewer for the 17th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2016, the 27th International Conference on Concurrency Theory CONCUR 2016, the 43rd International Colloquium on Automata, Languages and Programming 2016, the 31st ACM/IEEE Symposium on Logic in Computer Science, LICS 2016.

9.1.3. Journal

9.1.3.1. Member of the Editorial Boards

Jérôme Feret is a member of the editorial board of the Frontiers in Genetics journal and the Open Journal of Modeling and Simulation .

9.1.3.2. Reviewer - Reviewing Activities

Xavier Rival was a reviewer for ACM Transactions on Programming Languages and Systems TOPLAS.

Jerome Feret was a reviewer for Theoretical Computer Science 2016.

9.1.4. Invited Talks

Jérôme Feret gave “An overview of the Astrée/AstréeA analyzer. " at Journées scientifiques Inria Rennes, 20-22 june 2016 and at the workshop « Verified Trustworthy Software Systems » Imperial College, 6-7 April 2016.

Vincent Danos talked about ‘Residence: Simons Institute Program Logical Structures and Computations" at CONCUR 2016, Quebec, Aug 25-2 and at Berkeley, Aug 17-Dec 16. He also gave invited talks at SysMod SIG 2016, ISMB, Orlando, Jul 9, Xenobiology 2, XB2, May 24-26, Berlin, IPM Formal Methods Day, Teheran, Jan 10.

9.2. Teaching - Supervision - Juries

9.2.1. Teaching

Licence :

- Xavier Rival, “Semantics and Application to Verification",20h, Undergraduate course (L3), at Ecole Normale Supérieure
- Xavier Rival, “Introduction to Static Analysis", 8h, Course at Ecole des Mines de Paris, L3
- Cezara Drăgoi, “Programation concurrente et distribuée", Ecole Politechnique, L2
- Cezara Drăgoi, “Les principes des langages de programmation", Ecole Politechnique, L1
- Jérôme Feret, and Cezara Drăgoi, Mathematics, 40h, L1, FDV Bachelor program (Frontiers in Life Sciences (FdV)), Université Paris-Descartes, France.

Master :

- Vincent, Disruptive technologies and public policies, MSc Public affairs, Sciences Po, France.
- Xavier Rival, Protocol Safety and Verification, Master Course (M2) in the Advanced Communication Networks Master (12h hours), at Polytechnique and Ecole Nationale Supérieure des Telecoms
- Xavier Rival, “Verification" Lab Course at Ecole Polytechnique (M1, 20h)
- Vindent Danos and Jérôme Feret (with Jean Krivine), Computational Biology, 24h, M1. Interdisciplinary Approaches to Life Science (AIV), Master Program, Université Paris-Descartes, France.
- Cezara Drăgoi, Jérôme Feret, Antoine Miné, and Xavier Rival, Abstract Interpretation: application to verification and static analysis, 72h ETD, M2. Parisian Master of Research in Computer Science (MPRI). École normale supérieure. France.

Doctorat : Jérôme Feret, “Interprétation abstraite de modèles de voies de signalisation intracellulaire ", Lectures (3 hours) in the summer school “Modélisation Formelle de Réseaux de Régulation Biologique”, Porquerolles, June 2016 France.

9.2.2. Juries

Jérôme Feret was a member of the recruitment committee for an assistant professor in Paris-Diderot University 2016.

Vincent Danos was examiner and reviewer for the HDR of Sylvain Soliman (Ecole Polytechnique, 7th of December 2016).

10. Bibliography

Publications of the year

Articles in International Peer-Reviewed Journals

[1] W. ABOU-JAOUDÉ, D. THIEFFRY, J. FERET. *Formal Derivation of Qualitative Dynamical Models from Biochemical Networks*, in "BioSystems", September 2016, 100 p. [*DOI :* 10.1016/J.BIOSYSTEMS.2016.09.001], https://hal.inria.fr/hal-01379733

[2] J. LIU, X. RIVAL. *An array content static analysis based on non-contiguous partitions*, in "Computer Languages, Systems and Structures", 2017, vol. 47, n^{o} 1, pp. 104–129 [*DOI :* 10.1016/J.CL.2016.01.005], https://hal.inria.fr/hal-01399837

[3] A. OUADJAOUT, A. MINÉ, N. LASLA, N. BADACHE. *Static analysis by abstract interpretation of functional properties of device drivers in TinyOS*, in "Journal of Systems and Software", 2016, vol. 120, pp. 114–132 [*DOI :* 10.1016/J.JSS.2016.07.030], http://hal.upmc.fr/hal-01350646

International Conferences with Proceedings

[4] C. DRĂGOI, T. HENZINGER, D. ZUFFEREY. *PSYNC: A Partially Synchronous Language for Fault-Tolerant Distributed Algorithms*, in "POPL", Saint Petersburg , United States, January 2017 [*DOI :* 10.1145/NNNNNNN.NNNNNNN], https://hal.inria.fr/hal-01434325

[5] J. FERET, K. Q. LY. *Local traces: an over-approximation of the behaviour of the proteins in rule-based models*, in "CMSB 2016 - Fourteenth Conference on Computational Method in Systems Biology", Cambridge, United Kingdom, E. BARTOCCI, P. LIO', N. PAOLETTI (editors), Computational Methods in Systems Biology, Springer, September 2016, vol. 9859, pp. 116-131 [*DOI :* 10.1007/978-3-319-45177-0_8], https://hal.inria.fr/hal-01379897

[6] J. FERET, K. Q. LY. *Reachability analysis via orthogonal sets of patterns*, in "7th International Workshop on Static Analysis and Systems Biology, (SASB 2016)", Edinburgh, United Kingdom, D. SAFRANEK, G. SANGUINETTI (editors), Static Analysis and Systems Biology, Elsevier, September 2016, https://hal.inria.fr/hal-01379902

[7] T. SUZANNE, A. MINÉ. *From Array Domains to Abstract Interpretation Under Store-Buffer-Based Memory Models*, in "SAS 2016 - 23rd Static Analysis Symposium", Edinburgh, United Kingdom, Lecture Notes in Computer Science, Springer, September 2016, vol. 9837, pp. 469-488 [*DOI :* 10.1007/978-3-662-53413-7_23], http://hal.upmc.fr/hal-01360566

Conferences without Proceedings

[8] C. DRĂGOI, T. HENZINGER, D. ZUFFEREY. *PSYNC: A partially synchronous language for fault-tolerant distributed algorithms*, in "Proceedings of the 43nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages", Saint Petersburg, Florida, United States, January 2016 [*DOI :* 10.1145/NNNNNNN.NNNNNNN], https://hal.inria.fr/hal-01251199

[9] A. MINÉ, L. MAUBORGNE, X. RIVAL, J. FERET, P. COUSOT, D. KÄSTNER, S. WILHELM, C. FERDINAND. *Taking Static Analysis to the Next Level: Proving the Absence of Run-Time Errors and Data Races with Astrée*, in "8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)", Toulouse, France, January 2016, https://hal.archives-ouvertes.fr/hal-01271552

Books or Proceedings Editing

[10] X. RIVAL (editor). *Static Analysis: 23rd International Symposium, (SAS 2016), Edinburgh, UK, September 8-10, 2016, Proceedings*, Springer, Edinburgh, United Kingdom, 2016, vol. LNCS, n^{o} 9837 [*DOI :* 10.1007/978-3-662-53413-7], https://hal.inria.fr/hal-01388205

[11] C. ZHANG, X. RIVAL (editors). *State Of the Art in Program Analysis: International Workshop, (SOAP 2016), SOAP@PLDI 2016, Santa Barbara, CA, USA, June 14, 2016Proceedings of the 5th ACM SIGPLAN* , ACM, Santa Barbara, United States, 2016 [*DOI :* 10.1145/2931021], https://hal.inria.fr/hal-01388271

References in notes

[12] 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

[13] 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