IN PARTNERSHIP WITH:
CNRS

Ecole normale supérieure de Paris

Activity Report 2017

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
A2.1.9. - 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. - 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
B1.1.10. - Mathematical biology
B1.1.11. - Systems 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. Personnel

Research Scientists

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

Jerome Feret [Inria, Researcher]

Faculty Member

Patrick Cousot [New York University, Emeritus, until Nov 2017]

Post-Doctoral Fellow

Changhee Park [Inria, until Oct 2017]

PhD Students

Andreea Beica [Ecole Normale Supérieure Paris]
Marc Chevalier [Ecole Normale Supérieure Lyon, from Sep 2017]
Hugo Illous [Ecole Normale Supérieure Paris]
Huisong Li [Inria]
Thibault Suzanne [Ecole Normale Supérieure Paris]

Technical staff

Ferdinanda Camporesi [Inria]
Yves-Stan Le Cornec [Inria, from Sep 2017]
Jiangchao Liu [Inria, from Dec 2017]
Kim Quyen Ly [Inria]

Interns

Marc Chevalier [Ecole Normale Supérieure Lyon, until Aug 2017]
Guillaume Cluzel [Inria, from Jun 2017 until Aug 2017]
Sixiao Zhu [Inria, from Mar 2017 until Aug 2017]

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* [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-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 [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. 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 2017, with the proposal of a novel and groundbreaking way to improve the precision and scalability of analyses performed with disjunctive abstract domains, using silhouette abstraction.

5.1.1. Awards

Patrick Cousot received the IEEE John Von Neumann Medal.

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. OPENKAPPA

La platte-forme de modélisation OpenKappa
KEYWORDS: Model reduction - Simulation - Static analysis - Modeling - Systems Biology
SCIENTIFIC DESCRIPTION: OpenKappa is a collection of tools to build, debug and run models of biological pathways. It contains a compiler for the Kappa Language, a static analyzer (for debugging models), a simulator, a compression tool for causal traces, and a model reduction tool.

- 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

7. New Results

7.1. Memory Abstraction

7.1.1. Abstraction of arrays based on non contiguous partitions

Participants: Jiangchao Liu, Xavier Rival [correspondant].

In [9], 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.

7.1.2. Semantic-Directed Clumping of Disjunctive Abstract States

Participants: Huisong Li, Francois Berenger, Bor-Yuh Evan Chang, Xavier Rival [correspondant].

In [16], we studied the semantic directed clumping of disjunctive abstract states.

To infer complex structural invariants, Shape analyses rely on expressive families of logical properties. Many such analyses manipulate abstract memory states that consist of separating conjunctions of basic predicates describing atomic blocks or summaries. Moreover, they use finite disjunctions of abstract memory states in order to account for dissimilar shapes. Disjunctions should be kept small for the sake of scalability, though precision often requires to keep additional case splits. In this context, deciding when and how to merge case splits and to replace them with summaries is critical both for the precision and for the efficiency. Existing techniques use sets of syntactic rules, which are tedious to design and prone to failure. In this paper, we design a semantic criterion to clump abstract states based on their silhouette which applies not only to the conservative union of disjuncts, but also to the weakening of separating conjunction of memory predicates into inductive summaries. Our approach allows to define union and widening operators that aim at preserving the case splits that are required for the analysis to succeed. We implement this approach in the MemCAD analyzer, and evaluate it on real-world C codes from existing libraries, including programs dealing with doubly linked lists, red-black trees and AVL-trees.

7.1.3. Relational Inductive Shape Abstraction

Participants: Hugo Illous, Matthieu Lemerre, Xavier Rival [correspondant].

In [13], we studied a relational inductive shape abstract domain.

Static analyses aim at inferring semantic properties of programs. While many analyses compute an over-approximation of reachable states, some analyses compute a description of the input-output relations of programs. In the case of numeric programs, several analyses have been proposed that utilize relational numerical abstract domains to describe relations. On the other hand, designing abstractions for relations over memory states and taking shapes into account is challenging. In this paper, we propose 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 build an abstract domain and design a static analysis that over-approximates relations over memory states containing inductive structures. We implement this analysis and report on the analysis of a basic library of list manipulating functions.

7.2. Static Analysis of JavaScript Code

7.2.1. Weakly Sensitive Analysis for Unbounded Iteration over JavaScript Objects

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

In [14], 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.

7.2.2. Revisiting recency abstraction for JavaScript: towards an intuitive, compositional, and efficient heap abstraction

Participants: Jihyeok Park, Xavier Rival [correspondant], Sukyoung Ryu.

In [18], we studied recency abstractions and their use for the analysis of JavaScript programs.

JavaScript is one of the most widely used programming languages. To understand the behaviors of JavaScript programs and to detect possible errors in them, researchers have developed several static analyzers based on the abstract interpretation framework. However, JavaScript provides various language features that are difficult to analyze statically and precisely such as dynamic addition and removal of object properties, first-class property names, and higher-order functions. To alleviate the problem, JavaScript static analyzers often use recency abstraction, which refines address abstraction by distinguishing recent objects from summaries of old objects. We observed that while recency abstraction enables more precise analysis results by allowing strong updates on recent objects, it is not monotone in the sense that it does not preserve the precision relationship between the underlying address abstraction techniques: for an address abstraction A and a more precise abstraction B, recency abstraction on B may not be more precise than recency abstraction on A. Such an unintuitive semantics of recency abstraction makes its composition with various analysis sensitivity techniques also unintuitive. In this paper, we propose a new singleton abstraction technique, which distinguishes singleton objects to allow strong updates on them without changing a given address abstraction. We formally define recency and singleton abstractions, and explain the unintuitive behaviors of recency abstraction. Our preliminary experiments show promising results for singleton abstraction.

7.3. Astrée and AstréeA

7.3.1. Finding All Potential Run-Time Errors and Data Races in Automotive Software

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

Safety-critical embedded software has to satisfy stringent quality requirements. All contemporary safety standards require evidence that no data races and no critical run-time errors occur, such as invalid pointer accesses, buffer overflows, or arithmetic overflows. Such errors can cause software crashes, invalidate separation mechanisms in mixed-criticality software, and are a frequent cause of errors in concurrent and multi-core applications. The static analyzer ASTRÉE has been extended to soundly and automatically analyze concurrent software. This novel extension employs a scalable abstraction which covers all possible thread interleavings, and reports all potential run-time errors, data races, deadlocks, and lock/unlock problems. When the analyzer does not report any alarm, the program is proven free from those classes of errors. Dedicated support for ARINC 653 and OSEK/AUTOSAR enables a fully automatic OS-aware analysis. In [15], we give an overview of the key concepts of the concurrency analysis and report on experimental results obtained on concurrent automotive software. The experiments confirm that the novel analysis can be successfully applied to real automotive software projects.

7.4. Static analysis of signaling pathways

7.4.1. Formal and exact reduction for differential models of signaling pathways in rule-based languages

Participant: Ferdinanda Camporesi.

The behavior of a cell is driven by its capability to receive, propagate and communicate signals. Proteins can bind together on some binding sites. Post- translational modifications can reveal or hide some sites, so new interactions can be allowed or existing ones can be inhibited.

Due to the huge number of different bio-molecular complexes, we can no longer derive or integrate ODE models. A compact way to describe these systems is supplied by rule-based languages. However combinatorial complexity raises again when one attempt to describe formally the behavior of the models. This motivates the use of abstractions.

In this PhD thesis, we propose two methods to reduce the size of the models, that exploit respectively the presence of symmetries between sites and the lack of correlation between different parts of the system. The symmetries relates pairs of sites having the same capability of interactions. We show that this relation induces a bisimulation which can be used to reduce the size of the original model. The information flow analysis detects, for each site, which parts of the system influence its behavior. This allows us to cut the molecular species in smaller pieces and to write a new system. Moreover we show how this analysis can be tuned with respect to a context.

Both approaches can be combined. The analytical solution of the reduced model is the exact projection of the original one. The computation of the reduced model is performed at the level of rules, without the need of executing the original model.

7.4.2. Translating BNGL models into Kappa our experience

Participant: Kim Quyen Ly [correspondant].

So as to test the Kappa development tools on more examples, we translated the models provided with the BNGL distribution, into Kappa. In [20], we report about our experience. The translation was quite straightforward except for few interesting issues that we detail here. Firstly the use of static analysis has exposed some glitches in the modelling of some pathways in the models of the BNGL distribution. We explain how static analysis has helped us to detect, locate, and correct these flaws. Secondly, expanding BNGL rules using equivalent sites into rules with uniquely identified sites is not so easy when one wants to preserve faithfully the kinetics of interactions. We recall the semantics of BNGL for equivalent sites, and explain how to perform such translation.

7.4.3. Using alternated sums to express the occurrence number of extended patterns in site-graphs

Participants: Ferdinanda Camporesi, Jerome Feret [correspondant].

Site-graph rewriting languages as Kappa or BNGL supply a convenient way to describe models of signaling pathways. Unlike classical reaction networks, they emphasise on the biochemical structure of proteins. In [10], we use patterns to formalise properties about bio-molecular species. Intentionally, a pattern is a part of a species, but extensionally it denotes the multi-set of the species containing this pattern (with the multiplicity). Thus reasoning on patterns allows to handle symbolically arbitrarily big (if not infinite) multi-sets of species. This is a key point to design fast simulation algorithms or model reduction schemes. In this paper, we introduce the notion of extended patterns. Each extended pattern is made of a classical pattern and of a set of potential bonds between pairs of sites. Extended patterns have positive (when at least one of the potential bonds is realised) and negative (when none is realised) instances. They are important to express the consumption and the production of patterns by the rules that may break cycles in bio-molecular species by side-effects. We show that the number of positive (resp. negative) instances of extended patterns may be expressed as alternated sums of the number of occurrences of classical patterns.

7.4.4. KaDE: a Tool to Compile Kappa Rules into (Reduced) ODE Models

Participants: Ferdinanda Camporesi, Jerome Feret [correspondant], Kim Quyen Ly.

In [11], we introduce the tool KaDe, that may be used to compile models written in Kappa in ODE. Kappa is a formal language that can be used to model systems of biochemical interactions among proteins. It offers several semantics to describe the behaviour of Kappa models at different levels of abstraction. Each Kappa model is a set of context-free rewrite rules. One way to understand the semantics of a Kappa model is to read its rules as an implicit description of a (potentially infinite) reaction network. KaDE is interpreting this definition to compile Kappa models into reaction networks (or equivalently into sets of ordinary differential equations). KaDE uses a static analysis that identifies pairs of sites that are indistinguishable from the rules point of view, to infer backward and forward bisimulations, hence reducing the size of the underlying reaction networks without having to generate them explicitly. In [11], we describe the main current functionalities of KaDE and we give some benchmarks on case studies. A complete tutorial and more complete benchmarks may be found at the following url: http://www.di.ens.fr/~feret/CMSB2017-tool-paper/.

8. Bilateral Contracts and Grants with Industry

8.1. Bilateral Grants with Industry

Xavier Rival received a Facebook Faculty Award (2017).

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 - 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: Jerome 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. 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.

9.1.3. 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.

9.1.4. TGFSYSBIO

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

Type: ANR générique 2014

Defi: Société de l'information et de la communication

Instrument: Plan Cancer 2014-2019

Duration: December 2015 - November 2018

Coordinator: INSERM U1085-IRSET

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

Inria contact: Jerome 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.

9.2. European Initiatives

9.2.1. 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.

9.2.2. MemCad

Type: IDEAS

Defi: Design Composite Memory Abstract Domains

Instrument: ERC Starting Grant

Objectif: Design Composite Memory Abstract Domains

Duration: October 2011 - September 2016

Coordinator: Inria (France)

Partner: None

Inria contact: Xavier Rival

Abstract: The MemCAD project aims at setting up a library of abstract domains in order to express and infer complex memory properties. It is based on the abstract interpretation frameworks, which allows to combine simple abstract domains into complex, composite abstract domains and static analyzers. While other families of abstract domains (such as numeric abstract domains) can be easily combined (making the design of very powerful static analyses for numeric intensive applications possible), current tools for the analysis of programs manipulating complex abstract domains usually rely on a monolithic design, which makes their design harder, and limits their efficiency. The purpose of the MemCAD project is to overcome this limitation.

Our proposal is based on the observation that the complex memory properties that need to be reasoned about should be decomposed in combinations of simpler properties. Therefore, in static analysis, a complex memory abstract domain could be designed by combining many simpler domains, specific to common memory usage patterns. The benefit of this approach is twofold: first it would make it possible to simplify drastically the design of complex abstract domains required to reason about complex softwares, hereby allowing certification of complex memory intensive softwares by automatic static analysis; second, it would enable to split down and better control the cost of the analyses, thus significantly helping scalability. As part of this project, we propose to build a static analysis framework for reasoning about memory properties, and put it to work on important classes of applications, including large softwares.

9.3. International Initiatives

9.3.1. Participation in Other International Programs

9.3.1.1. EXEcutable Knowledge

Title: EXEcutable Knowledge

Type: DARPA

Instrument: DARPA Program

Program: Big Mechanism

Duration: July 2014 - December 2017

Coordinator: Harvard Medical School (Boston, USA)

Partner: Inria Paris-Rocquencourt, École normale supérieure de Lyon Université Paris-Diderot,

Inria contact: Jerome Feret

Abstract: Our overarching objective is Executable Knowledge: to make modeling and knowledge representation twin sides of biological reasoning. This requires the definition of a formal language with a clear operational semantics for representing proteins and their interaction capabilities in terms of agents and rules informed by, but not exposing, biochemical and biophysical detail. Yet, to achieve Executable Knowledge we need to go further:

- Bridge the gap between rich data and their formal representation as executable model elements. Specifically, we seek an intermediate, but already formal, knowledge representation (meta-language) to express granular data germane to interaction mechanisms; a protocol defining which and how data are to be expressed in that language; and a translation procedure from it into the executable format.
- Implement mathematically sound, fast, and scalable tools for analyzing and executing arbitrary collections of rules.
- Develop a theory of causality and attendant tools to extract and analyze the unfolding of causal lineages to observations in model simulations.

We drive these technical goals with the biological objective of assembling rule-based models germane to Wnt signaling in order to understand the role of combinatorial complexity in robustness and control.

9.3.1.2. *Active Context*

Title: Active Context

Type: DARPA

Instrument: DARPA Program

Program: Communicating with Computers

Duration: July 2015 - December 2018

Coordinator: Harvard Medical School (Boston, USA)

Partner: University of California, (San Diego, USA), Inria Paris-Rocquencourt, École normale supérieure de Lyon Université Paris-Diderot,

Inria contact: Jerome Feret

Abstract: The traditional approach to the curation of biological information follows a philatelic paradigm, in which epistemic units based on raw or processed data are sorted, compared and catalogued in a slow and all too often insufficiently coordinated process aimed at capturing the meaning of each specimen in isolation. The swelling bounty of data generated by a systematic approach to biology founded on high-throughput technologies appears to have only intensified a sense of disconnected facts, despite their rendering as networks. This is all the more frustrating as the tide of static data (sequences, structures) is giving way to a tide of dynamic data about (protein-protein) interaction that want to be interconnected and understood (think annotated) in terms of process, i.e. a systemic approach.

The barrier is the complexity of studying systems of numerous heterogeneously interacting components in a rapidly evolving field of science. The complexity comes from two kinds of dynamically changing context: the internal dynamics of a biological system, which provide the context for assessing the meaning of a protein-protein interaction datum, and the external dynamics of the very fact base used to define the system in the first place. We propose the integration of dynamic modeling into the practice of bioinformatics to address these two dynamics by coupling them. The external dynamics is at first handled by a novel kind of two-layered knowledge representation (KR). One layer contextualizes proteins and their interactions in a structure that incrementally constructs, in an open-ended dialogue with the user, its own semantics by piecing together fragments of knowledge from a variety of sources tapped by the Big Mechanism program. The other layer is a model representation (MR) that handles and prioritizes the many executable abstractions compatible with the KR. The internal dynamics is handled not only by execution but also by addressing the impedance mismatch between the unwieldy formal language(s) required for execution and the more heuristic, high-level concepts that structure the modeling discourse with which biologists reason about molecular signaling systems. To the extent that we are successful on both ends, users will be able to effectively deploy modeling for curating the very fact base it rests upon, hopefully achieving self-consistency.

9.4. International Research Visitors

9.4.1. Visits of International Scientists

9.4.1.1. *Internships*

Xavier Rival supervised the internship of Guillaume Cluzel (L3, École Normale Supérieure de Lyon), on the implementation of array abstract domains.

Xavier Rival supervised the internship of Sixiao Zhu (M1, École Polytechnique), on the integration of a three valued abstraction in MemCAD.

9.4.2. Visits to International Teams

9.4.2.1. Research Stays Abroad

Xavier Rival visited KAIST (Korean Advanced Institute for Science and Technology) as an Invited Professor in November/December 2017.

10. Dissemination

10.1. Promoting Scientific Activities

10.1.1. Scientific Events Selection

10.1.1.1. Chair of Conference Program Committees

- Jerome Feret served as co-Chair of CMSB 2017 (Computational Methods in Systems Biology).
- Xavier Rival is serving as Chair of the Artifact Evaluation Committee of SAS 2018 (Static Analysis Symposium).

10.1.1.2. Member of the Conference Program Committees

- Jerome Feret served as a Member of the Program Committee of TMPA 2017 (Tools and Methods of Program Analysis).
- Jerome Feret served as a Member of the Program Committee of JOBIM 2017 (Journées Ouvertes en Biologie, Informatique et Mathématiques).
- Jerome Feret served as a Member of the Program Committee of SASB 2017 (Static Analysis and System Biology).
- Jerome Feret is serving as a Member of the Program Committee of LICS 2018 (Logic in Computer Science).
- Jerome Feret is serving as a Member of the Program Committee of VEMDP 2018 (Verification of Engineered Molecular Devices and Programs).
- Jerome Feret is serving as a Member of the Program Committee of SAS 2018 (Static Analysis Symposium).
- Jerome Feret is serving as a Member of the Program Committee of CMSB 2018 (Computational Methods in Systems Biology).
- Xavier Rival was a Member of the Program Committee of SAS 2017 (Static Analysis Symposium).
- Xavier Rival is serving as a Member of the Program Committee of SAS 2018 (Static Analysis Symposium).
- Xavier Rival was a Member of the Program Committee of Web Design, Analysis, Programming and Implementation of the WWW'18 Conference.
- Xavier Rival was a Member of the Extended Review Committee of PLDI 2018 (Programming Languages Design and Implementation).
- Cezara Dragoi was a member of Programming languages design and implementation PLDI'17.
- Cezara Dragoi was a member of Computer Aided Verification CAV'18.

10.1.1.3. Reviewer

- Jerome Feret served as reviewer for CONCUR 2017 (Concurrency Theory).
- Jerome Feret served as reviewer for LICS 2017 (Logic in Computer Science).
- Jerome Feret served as reviewer for VMCAI 2017 (Verification, Model Checking, and Abstract Interpretation).

10.1.2. Journal

10.1.2.1. Member of the Editorial Boards

- Jerome Feret is a member of the editorial board of the Frontiers in Genetics journal and the Open Journal of Modeling and Simulation.
- Jerome Feret serves as co-Editor of an Issue of the Theoretical Computer Science journal, that is composed of papers from SASB 2016, and is expected to appear in 2018.
- Jerome Feret serves as Editor of an Issue of the IEEE/ACM Transactions on Computational Biology and Bioinformatics, that is composed of papers from CMSB 2016, and is expected to appear in 2019.
- Xavier Rival serves as Editor of an Issue of the Formal Methods in System Design Journal, that is composed of a selection of papers from SAS 2016, and is expected to appear in 2018.

10.1.2.2. Reviewer - Reviewing Activities

- Jerome Feret served as a Reviewer for FMSD (Formal Methods in System Design).
- Jerome Feret served as a Reviewer for TCS (Theoretical Computer Sciences).
- Jerome Feret served as a Reviewer for TCBB (IEEE/ACM Transactions on Computational Biology and Bioinformatics).
- Jerome Feret served as a Reviewer for TCL (Transactions on Computational Logic).
- Xavier Rival served as a Reviewer for ACM TOPLAS (Transactions On Programming Languages and Systems).
- Xavier Rival served as a Reviewer for ACM TOPS (Transactions On Privacy and Security).

10.1.3. Invited Talks

- Jerome Feret gave a talk on automatic reduction of models of intra-cellular signaling pathways at the Dagstuhl Seminar on Algorithmic Cheminformatics (5–10 November 2017).
- Cezara Dragoi was invited speaker at the 6th South of England Regional Programming Language Seminar University College London, Workshop on Software Correctness and Reliability at ETH Zurich, and Workshop on Formal Reasoning in Distributed Algorithms (FRIDA), Wien Austria.

10.1.4. Leadership within the Scientific Community

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

10.1.5. Research Administration

Jerome Feret and Xavier Rival are members of the Laboratory Council of DIENS.

10.2. Teaching - Supervision - Juries

10.2.1. Teaching

Licence:

- Jerome Feret, and Marc Chevalier, Mathematics, 40h, L1, FDV Bachelor program (Frontiers in Life Sciences (FdV)), Université Paris-Descartes, France.
- Jerome Feret and Xavier Rival, “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 “Programmation Avancée”, 18h, L3, at École Polytechnique, France.
- Cezara Dragoi “Les principes des langages de programmation”, 18h, L1, at École Polytechnique, France.

Master:

- Xavier Rival, “Verification” Lab Course, 20h, M1, École Polytechnique, France.
- Xavier Rival, “Protocol Safety and Verification”, 12h, M2, Advanced Communication Networks Master, France.
- Xavier Rival, “Program Analysis”, 24h, M2, Korea Advanced Institute for Science and Technology (KAIST), South Korea.
- Cezara Drăgoi, Jerome 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 Jerome Feret (with Jean Krivine), Computational Biology, 28h, M1. Interdisciplinary Approaches to Life Science (AIV), Master Program, Université Paris-Descartes, France.

10.2.2. Supervision

- PhD defended: Ferdinanda Camporesi, Formal and exact reduction for differential models of signaling pathways in rule-based languages. Defended the 23th of January, 2017 and supervised by Jerome Feret.
- PhD in progress: Marc Chevalier, Static analysis of Security Properties in Critical Embedded Software. started in 2017 and supervised by Jerome Feret
- PhD in progress: Hugo Illous, Relational Shape Abstraction Based on Separation Logic, started in 2015 and supervised by Xavier Rival and Matthieu Lemerre (CEA)
- PhD in progress: Huisong Li, Disjunctive Shape Abstraction for Shared Data-Structures, started in 2014 and supervised by Xavier Rival
- PhD in progress: Jiangchao Liu, Static Analysis for Numeric and Structural Properties of Array Contents, started in 2014 and supervised by Xavier Rival

10.2.3. Juries

- Jerome Feret served as a member of the Jury of the PhD of Jean Coquet (Defended the 20th of December, 2017).
- Xavier Rival served as a Reviewer in the Jury of the PhD of Ahmad Salim Al-Sibahi (Defense planned for the 11th of January, 2018).

10.2.4. Responabilities

- Jerome Feret is a member of the Monotoring Committee for PhD Studies (CSD) of Inria Paris.
- Jerome Feret is deputy head of studies of the Computer Science department of École normale supérieure.

10.2.5. Selection committees

- Jerome Feret was a member of the recruitment committee for an assistant professor in Evry University 2017.
- Jerome Feret is a member of the recruitment committee for an assistant professor in Paris-Diderot University 2018.

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. DRĂGOI, 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

Articles in International Peer-Reviewed Journals

[9] 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

International Conferences with Proceedings

[10] F. CAMPORESI, J. FERET. *Using alternated sums to express the occurrence number of extended patterns in site-graphs*, in "SASB 2017 - The Eighth International Workshop on Static Analysis for Systems Biology", New York, United States, J. YANG, J. A. BACHMAN (editors), Static Analysis and Systems Biology, Elsevier, August 2017, 18 p. , To appear, https://hal.inria.fr/hal-01613603

[11] F. CAMPORESI, J. FERET, K. Q. LY. *KaDE: A Tool to Compile Kappa Rules into (Reduced) ODE Models*, in "CMSB 2017 - 15th Conference on Computational Methods in Systems Biology", Darmstadt, Germany, J. FERET, H. KOEPPL (editors), Computational Methods in Systems Biology, Springer, September 2017, vol. 10545, pp. 291-299, Tools paper track [*DOI :* 10.1007/978-3-319-67471-1_18], https://hal.inria.fr/hal-01613600

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

[13] H. ILLOUS, M. LEMERRE, X. RIVAL. *A Relational Shape Abstract Domain*, in "NFM 2017 - 9th NASA Formal Methods Symposium", Moffett Field, United States, LNCS, Springer, April 2017, vol. 10227, pp. 212-229 [*DOI :* 10.1007/978-3-319-57288-8_15], https://hal.inria.fr/hal-01648681

[14] 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

[15] D. KÄSTNER, A. MINÉ, A. SCHMIDT, H. HILLE, L. MAUBORGNE, S. WILHELM, X. RIVAL, J. FERET, P. COUSOT, C. FERDINAND. *Finding All Potential Run-Time Errors and Data Races in Automotive Software*, in "SAE world Congress", Detroit , United States, SAE Technical Paper, SAE International, April 2017, 9 p. , https://hal.inria.fr/hal-01674831

[16] H. LI, F. BÉRENGER, B.-Y. E. CHANG, X. RIVAL. *Semantic-Directed Clumping of Disjunctive Abstract States* *, in "POPL 2017 - 44th ACM SIGPLAN Symposium on Principles of Programming Languages", Paris, France, ACM, January 2017, vol. 52, n^{o} 1, pp. 32-45 [*DOI :* 10.1145/3009837.3009881], https://hal.inria.fr/hal-01648679

[17] R. MONAT, A. MINÉ. *Precise Thread-Modular Abstract Interpretation of Concurrent Programs Using Relational Interference Abstractions*, in "Verification, Model Checking, and Abstract Interpretation (VMCAI) 2017", Paris, France, A. BOUAJJANI, D. MONNIAUX (editors), Lecture Notes in Computer Science, Springer, January 2017, vol. 10145, pp. 386-404 [*DOI :* 10.1007/978-3-319-52234-0_21], https://hal.inria.fr/hal-01490178

[18] J. PARK, X. RIVAL, S. RYU. *Revisiting Recency Abstraction for JavaScript Towards an Intuitive, Compositional, and Efficient Heap Abstraction*, in "SOAP 2017 - International Workshop on the State Of the Art in Java Program Analysis", Barcelona, Spain, June 2017, pp. 1-6 [*DOI :* 10.1145/3088515.3088516], https://hal.inria.fr/hal-01648682

Books or Proceedings Editing

[19] J. FERET, H. KOEPPL (editors). *Computational Methods in Systems Biology : 15th International Conference, CMSB 2017, Darmstadt, Germany, September 27–29, 2017, Proceedings*, Lecture Notes in Bioinformatics, Springer, France, September 2017, vol. 10545, 332 p. [*DOI :* 10.1007/978-3-319-67471-1], https://hal.inria.fr/hal-01613596

Other Publications

[20] K. Q. LY. *Translating BNGL models into Kappa our experience*, August 2017, 4 p. , Extended abstract, https://hal.inria.fr/hal-01613604

References in notes

[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