IN PARTNERSHIP WITH:
CNRS

Ecole normale supérieure de Paris

Activity Report 2015

Project-Team ANTIQUE

Static Analysis by Abstract Interpretation

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

RESEARCH CENTER
Paris - Rocquencourt

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. - Reliability, certification
2.4.1. - Analysis
2.4.2. - Verification
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, Researcher, HdR]
Patrick Cousot [ENS Paris, Professor Emeritus]
Vincent Danos [CNRS, HdR]
Cezara Drăgoi [Inria, Researcher, from Feb 2015]
Jérôme Feret [Inria, Researcher]
Antoine Miné [CNRS, Researcher, until Aug 2015, HdR]

Engineers

Francois Berenger [Inria]
Tie Cheng [Inria]
Kim Quyen Ly [Inria]

PhD Students
Mehdi Bouaziz [ENS Paris]
Ferdinanda Camporesi [ENS Paris]
Huisong Li [Inria]
Illous Hugo [CEA]
Thibault Suzanne [ENS Paris]
Antoine Toubhans [Inria, PhD student, until Aug 2015]
Caterina Urban [ENS Paris, until Aug 2015]

Post-Doctoral Fellow
Ilias Garnier [ENS Paris]

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* [31]. 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* [31], 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 [30], 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. New Software and Platforms

5.1. 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/

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

5.3. 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/

5.4. 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

5.5. 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

5.6. 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

5.7. 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

5.8. 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/

5.9. 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

5.10. 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

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

5.12. 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. New Results

6.1. Memory Abstraction

6.1.1. Abstraction of arrays based on non contiguous partitions

Participants: Jiangchao Liu, Xavier Rival [correspondant].

Abstract interpretation, Memory abstraction, Array abstract domains. In [19], 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.

6.1.2. Static analysis for unstructured sharing

Participants: Huisong Li, Bor-Yuh Evan Chang [University of Colorado, Boulder, USA], Xavier Rival [correspondant].

Abstract interpretation, Memory abstraction, Separation logic. In [18], we studied the abstraction of shared data-structures.

Shape analysis aims to infer precise structural properties of imperative memory states and has been applied heavily to verify safety properties on imperative code over pointer-based data structures. Recent advances in shape analysis based on separation logic has leveraged summarization predicates that describe unbounded heap regions like lists or trees using inductive definitions. Unfortunately, data structures with *unstructured sharing*, such as graphs, are challenging to describe and reason about in such frameworks. In particular, when the sharing is unstructured, it cannot be described inductively in a local manner. In this work, we proposed a global abstraction of sharing based on set-valued variables that when integrated with inductive definitions enables the specification and shape analysis of structures with unstructured sharing.

6.1.3. Synthesizing short-circuiting validation of data structure invariants

Participants: Yi-Fan Tsai, Devin Coughlin, Bor-Yuh Evan Chang [University of Colorado, Boulder, USA], Xavier Rival [correspondant].

In [28], we studied the synthesis of short-circuiting validators for data-structure invariants.

This work introduces *incremental verification-validation*, a novel approach for checking rich data structure invariants expressed as separation logic assertions. Incremental verification-validation combines static verification of separation properties with efficient, *short-circuiting* dynamic validation of arbitrarily rich data constraints. A data structure invariant checker is an inductive predicate in separation logic with an executable interpretation; a short-circuiting checker is an invariant checker that stops checking whenever it detects at *run time* that an assertion for some sub-structure has been fully proven *statically*. At a high level, our approach does two things: it statically proves the separation properties of data structure invariants using a static shape analysis in a standard way but then leverages this proof in a novel manner to synthesize short-circuiting dynamic validation of the data properties. As a consequence, this approach enables dynamic validation to make up for imprecision in sound static analysis while simultaneously leveraging the static verification to make the remaining dynamic validation efficient. This work has shown empirically that short-circuiting can yield asymptotic improvements in dynamic validation, with low overhead over no validation, even in cases where static verification is incomplete.

6.2. Abstract domains

6.2.1. Abstract domains and solvers for set reasoning

Participants: Arlen Cox, Bor-Yuh Evan Chang [University of Colorado, Boulder, USA], Huisong Li, Xavier Rival [correspondant].

In [15], we studied the abstraction and inferrence of set properties.

When constructing complex program analyses, it is often useful to reason about not just individual values, but collections of values. Symbolic set abstractions provide building blocks that can be used to partition elements, relate partitions to other partitions, and determine the provenance of multiple values, all without knowing any concrete values. To address the simultaneous challenges of scalability and precision, we formalized and implemented an interface for symbolic set abstractions and constructed multiple abstract domains relying on both specialized data structures and off-the-shelf theorem provers. We developed techniques for lifting existing domains to improve performance and precision. We evaluated these domains on real-world data structure analysis problems.

6.2.2. Abstraction of optional numerical values

Participants: Jiangchao Liu, Xavier Rival [correspondant].

In [20], we designed a functor to lift a numerical abstract domain into an abstract domain that accounts for *optional* numerical values.

We proposed a technique to describe properties of numerical stores with optional values, that is, where some variables may have no value. Properties of interest include numerical equalities and inequalities. Our approach lifts common linear inequality based numerical abstract domains into abstract domains describing stores with optional values. This abstraction can be used in order to analyze languages with some form of *option* scalar type. It can also be applied to the construction of abstract domains to describe complex memory properties that introduce symbolic variables, e.g., in order to summarize unbounded sets of program variables, and where these symbolic variables may be undefined, as in some array or shape analyses. We described the general form of abstract states, and propose sound and automatic static analysis algorithms. We evaluated our construction in the case of an array abstract domain.

6.3. Static analysis of JavaScript applications

6.3.1. Desynchronized multi-state abstractions for open programs in dynamic languages

Participants: Arlen Cox [correspondant], Bor-Yuh Evan Chang [University of Colorado, Boulder, USA], Xavier Rival.

Abstract interpretation, Dynamically typed languages, Verification In [16], we have studied desynchronized multi-state abstractions for open programs in dynamic languages (libraries).

Dynamic language library developers face a challenging problem: ensuring that their libraries will behave correctly for a wide variety of client programs without having access to those client programs. This problem stems from the common use of two defining features for dynamic languages: callbacks into client code and complex manipulation of attribute names within objects. To remedy this problem, we introduced two state-spanning abstractions. To analyze callbacks, the first abstraction desynchronizes a heap, allowing partitions of the heap that may be affected by a callback to an unknown function to be frozen in the state prior to the call. To analyze object attribute manipulation, building upon an abstraction for dynamic language heaps, the second abstraction tracks attribute name/value pairs across the execution of a library. We implemented these abstractions and use them to verify modular specifications of class-, trait-, and mixin-implementing libraries.

6.4. Static analysis of spreadsheet applications

Participants: Tie Cheng [correspondant], Xavier Rival.

Abstract interpretation, Spreadsheet applications, Verification In [14], we have proposed a static analysis to detect type unsafe operations in spreadsheet applications including formulas and macros.

Spreadsheets are widely used, yet are error-prone: they use a weak type system, allowing certain operations that will silently return unexpected results, like comparisons of integer values with string values. However, discovering these issues is hard, since data and formulas can be dynamically set, read or modified. We defined a static analysis that detects all run-time type-unsafe operations in spreadsheets. It is based on an abstract interpretation of spreadsheet applications, including spreadsheet tables, global re-evaluation and associated programs. Our implementation supports the features commonly found in real-world spreadsheets. We ran our analyzer on the EUSES Spreadsheet Corpus. This evaluation shows that our tool is able to automatically verify a large number of real spreadsheets, runs in a reasonable time and discovers complex bugs that are difficult to detect by code review or by testing.

6.5. Distributed systems verification and programming language

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 notoriously difficult to implement correctly, due to asynchronous communication and the occurrence of faults, such as the network dropping messages or computers crashing. Noteworthy is the lack of automated verification techniques for distributed systems, highly contrasting the mass distribution and development of distributed software. Therefore, our main motivation is to increase the confidence we have in distributed systems using formal verification methods. However, 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. We think that the difficulty does not only come from the algorithms but from the way we think about distributed systems. Therefore, we are interested in finding an appropriate programming model for fault-tolerant distributed algorithms, that increases the confidence we have distributed software. We introduced PSYNC, a domain specific language based on the Heard-Of model, which views asynchronous faulty systems as synchronous ones with an adversarial environment that simulates asynchrony and faults by dropping messages. We defined a runtime system for PSYNC that efficiently executes on asynchronous networks. We formalize the relation between the runtime system and PSYNC in terms of observational refinement. PSYNC introduces a high-level lockstep abstraction (on top of the standard asynchronous semantics), which simplifies the design and implementation of fault-tolerant distributed algorithms and enables automated formal verification. We have implemented an embedding of PSYNC in the SCALA programming language with a runtime system for asynchronous networks. We showed the applicability of PSYNC by implementing several important fault-tolerant distributed algorithms and we compared the implementation of consensus algorithms in PSYNC against implementations in other languages in terms of code size, runtime efficiency, and verification.

6.6. Derivation of Qualitative Dynamical Models from Biochemical Networks

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

Systems biology, Logical models, Automatic derivation 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 [13] a methodology for the derivation of qualitative dynamical models from biochemical networks. Properties are formalized 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.

6.7. Annotation of rule-based models with formal semantics to enable creation, analysis, reuse and visualization

Participants: G. Misirli, M. Cavaliere, W. Waites, M. Pocock, C. Madsen, O. Gifellon, R. Honorato-Zimmer, P. Zuliani, V. Danos [correspondant], A. Wipat.

In [35] We present an annotation framework and guidelines for annotating rule-based models, encoded in the commonly used Kappa and BioNetGen languages. Biological systems are complex and challenging to model and therefore model reuse is highly desirable. To promote model reuse, models should include both information about the specifics of simulations and the underlying biology in the form of metadata. The availability of computationally tractable metadata is especially important for the effective automated interpretation and processing of models. Metadata are typically represented as machine-readable annotations which enhance programmatic access to information about models. Rule-based languages have emerged as a modelling framework to represent the complexity of biological systems. Annotation approaches have been widely used for reaction-based formalisms such as SBML. However, rule-based languages still lack a rich annotation framework to add semantic information, such as machine-readable descriptions, to the components of a model. We introduced an annotation framework and guidelines for annotating rule-based models, encoded in the commonly used Kappa and BioNetGen languages. We adapted widely adopted annotation approaches to rule-based models. We initially proposed a syntax to store machine-readable annotations and describe a mapping between rule-based modelling entities, such as agents and rules, and their annotations. We then described an ontology to both annotate these models and capture the information contained therein, and demonstrate annotating these models using examples. Finally, we presented a proof of concept tool for extracting annotations from a model that can be queried and analyzed in a uniform way. The uniform representation of the annotations can be used to facilitate the creation, analysis, reuse and visualization of rule-based models. Although examples are given, using specific implementations the proposed techniques can be applied to rule-based models in general.

6.8. Quantitative genomic analysis of RecA protein binding during DNA double-strand break repair reveals RecBCD action in vivo

Participants: Charlotte Cockram, Milana Filatenkova, Vincent Danos [correspondant], Meriem Karoui, Leach David.

Understanding molecular mechanisms in the context of living cells requires the development of new methods of in vivo biochemical analysis to complement established in vitro biochemistry. A critically important molecular mechanism is genetic recombination, required for the beneficial reassortment of genetic information and for DNA double-strand break repair (DSBR). Central to recombination is the RecA (Rad51) protein that assembles into a spiral filament on DNA and mediates genetic exchange. Here we developed a method that combines chromatin immunoprecipitation with next-generation sequencing (ChIP-Seq) and mathematical modeling to quantify RecA protein binding during the active repair of a single DSB in the chromosome of Escherichia coli. In [29] we have used quantitative genomic analysis to infer the key in vivo molecular parameters governing RecA loading by the helicase/ nuclease RecBCD at recombination hot-spots, known as Chi. Our genomic analysis has also revealed that DSBR at the lacZ locus causes a second RecBCD-mediated DSBR event to occur in the ter- minus region of the chromosome, over 1 Mb away.

6.9. Moment Semantics for Reversible Rule-Based Systems

Participants: Vincent Danos [correspondant], Tobias Hinder, Ricardo Honorato-Zimmer, Sandro Stuck.

In [34] we developed a notion of stochastic rewriting over marked graphs – i.e. directed multigraphs with degree constraints. The approach is based on double-pushout (DPO) graph rewriting. Marked graphs are expressive enough to internalize the ‘no-dangling-edge’ condition inherent in DPO rewriting. Our main result is that the linear span of marked graph occurrence-counting functions – or motif functions – form an algebra which is closed under the infinitesimal generator of (the Markov chain associated with) any such rewriting

system. This gives a general procedure to derive the moment semantics of any such rewriting system, as a countable (and recursively enumerable) system of differential equations indexed by motif functions. The differential system describes the time evolution of moments (of any order) of these motif functions under the rewriting system. We illustrate the semantics using the example of preferential attachment networks; a well-studied complex system, which meshes well with our notion of marked graph rewriting. We show how in this case our procedure obtains a finite description of all moments of degree counts for a fixed degree.

6.10. Dirichlet is Natural

Participants: Vincent Danos [correspondant], Ilias Garnier.

In [32] the authors reconstruct a family of higher-order probabilities known as the Dirichlet process.

Giry and Lawvere's categorical treatment of probabilities, based on the probabilistic monad G, offer an elegant and hitherto unexploited treatment of higher-order probabilities. The goal of this paper is to follow this formulation to reconstruct a family of higher-order probabilities known as the Dirichlet process. This family is widely used in non-parametric Bayesian learning.

Given a Polish space X, we build a family of higher-order probabilities in G(G(X)) indexed by M(X), the set of non-zero finite measures over X. The construction relies on two ingredients. First, we develop a method to map a zero-dimensional Polish space X to a projective system of finite approximations, the limit of which is a zero-dimensional compactification of X. Second, we use a functorial version of Bochner's probability extension theorem adapted to Polish spaces, where consistent systems of probabilities over a projective system give rise to an actual probability on the limit. These ingredients are combined with known combinatorial properties of Dirichlet processes on finite spaces to obtain the Dirichlet family on X. We prove that the Dirichlet family is a natural transformation from the monad M to GG over Polish spaces, which in particular is continuous in its parameters. This is an improvement on extant constructions of Dirichlet.

6.11. Mechanistic links between cellular trade-offs, gene expression, and growth

Participants: Andrea Weisse, Diego Oyarzun, Vincent Danos [correspondant], Peter Swain.

Intracellular processes rarely work in isolation but continually interact with the rest of the cell. In microbes, for example, we now know that gene expression across the whole genome typically changes with growth rate. The mechanisms driving such global regulation, however, are not well understood. In [36] we considered three trade-offs that, because of limitations in levels of cellular energy, free ribosomes, and proteins, are faced by all living cells and we construct a mechanistic model that comprises these trade-offs. Our model couples gene expression with growth rate and growth rate with a growing population of cells. We show that the model recovers Monod's law for the growth of microbes and two other empirical relationships connecting growth rate to the mass fraction of ribosomes. Further, we can explain growth-related effects in dosage compensation by paralogs and predict host–circuit interactions in synthetic biology. Simulating competitions between strains, we find that the regulation of metabolic pathways may have evolved not to match expression of enzymes to levels of extracellular substrates in changing environments but rather to balance a trade-off between exploiting one type of nutrient over another. Although coarse-grained, the trade-offs that the model embodies are fundamental, and, as such, our modeling framework has potentially wide application, including in both biotechnology and medicine.

6.12. Thermodynamic graph-rewriting

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

In [33] we developped a new thermodynamic approach to stochastic graph-rewriting. The ingredients are a finite set of reversible graph-rewriting rules called generating rules, a finite set of connected graphs P called energy patterns and an energy cost function. The idea is that the generators define the qualitative dynamics, by showing which transformations are possible, while the energy patterns and cost function specify the long-term probability π of any reachable graph. Given the generators and energy patterns, we construct a finite set of rules which (i) has the same qualitative transition system as the generators; and (ii) when equipped with suitable rates, defines a continuous-time Markov chain of which π is the unique fixed point. The construction relies on the use of site graphs and a technique of 'growth policy' for quantitative rule refinement which is of independent interest. This division of labour between the qualitative and long-term quantitative aspects of the dynamics leads to intuitive and concise descriptions for realistic models (see the examples in S4 and S5). It also guarantees thermodynamical consistency (AKA detailed balance), otherwise known to be undecidable, which is important for some applications. Finally, it leads to parsimonious parameterizations of models, again an important point in some applications.

6.13. Kappa Rule-Based Modelling in Synthetic Biology

Participants: John Wilson-Kanamori, Vincent Danos [correspondant], Ty Thomson, Ricardo Honorato-Zimmer.

This [37] is a chapter of a book that provides complete coverage of the computational approaches currently used in Synthetic Biology. Rule-based modeling, an alternative to traditional reaction-based modeling, allows us to intuitively specify biological interactions while abstracting from the underlying combinatorial complexity. One such rule-based modeling formalism is Kappa, which we introduce to readers in this chapter. We discuss the application of Kappa to three modeling scenarios in synthetic biology: a unidirectional switch based on nitrosylase induction in Saccharomyces cerevisiae, the repressilator in Escherichia coli formed from BioBrick parts, and a light-mediated extension to said repressilator developed by the University of Edinburgh team during iGEM 2010. The second and third scenarios in particular form a case-based introduction to the Kappa BioBrick Framework, allowing us to systematically address the modeling of devices and circuits based on BioBrick parts in Kappa. Through the use of these examples, we highlight the ease with which Kappa can model biological interactions both at the genetic and the protein–protein interaction level, resulting in detailed stochastic models accounting naturally for transcriptional and translational resource usage. We also hope to impart the intuitively modular nature of the modeling processes involved, supported by the introduction of visual representations of Kappa models. Concluding, we explore future endeavors aimed at making modeling of synthetic biology more user-friendly and accessible, taking advantage of the strengths of rule-based modeling in Kappa.

This Chapters focus on computational methods and algorithms for the design of bio-components, insight on CAD programs, analysis techniques, and distributed systems. Written in the highly successful Methods in Molecular Biology series format, the chapters include the kind of detailed description and implementation advice that is crucial for getting optimal results in the laboratory.

Authoritative and practical, Computational Methods in Synthetic Biology serves as a guide to plan in silico the in vivo or in vitro construction of a variety of synthetic bio-circuits.

7. Partnerships and Cooperations

7.1. National Initiatives

7.1.1. ANR

7.1.1.1. 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.

7.1.1.2. VerAsCo

Title: Formally-verified static analyzers and compilers

Type: ANR Ingénierie Numérique Sécurité 2011

Instrument: ANR grant

Duration: Septembre 2011 - September 2015

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.

7.1.1.3. AstréeA

Title: Static Analysis of Embedded Asynchronous Real-Time Software

Type: ANR Ingénierie Numérique Sécurité 2011

Instrument: ANR grant

Duration: January 2012 - December 2015

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.

7.2. European Initiatives

7.2.1. FP7 & H2020 Projects

7.2.1.1. 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)

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.

7.3. International Initiatives

7.3.1. 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: Jérôme 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.

7.3.2. 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: Jérôme 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.

7.4. International Research Visitors

7.4.1. Visits of International Scientists

Josef Widder, associate professor at TU Wien, Embedded Computing Systems group, visited Cezara Drăgoi for a week, from Oct 12 to Oct 17.

7.4.1.1. Internships

Jérôme Feret is supervizing the Internships of Ken Chanseau Germain (M2 student), on “approximated model reduction of differential semantics”, since november 2015.

8. Dissemination

8.1. Promoting Scientific Activities

8.1.1. Scientific events organisation

8.1.1.1. Member of steering committees

Jérôme Feret is member of the Steering Committee of the international Workshop on Static Analysis and Systems Biology (SASB). Xavier Rival is member of the Steering Committee of the international workshop on Tools for Automatic Program Analysis (TAPAS).

8.1.2. Scientific events selection

8.1.2.1. Chair of conference program committees

Xavier Rival is Co-Chair of the Program Committee of the Workshop on State Of the Art Program Analysis (SOAP'16), to be held in Santa Barbara (USA), June 2016. Xavier Rival is the Chair of the Program Committee of the 23rd Static Analysis Symposium (SAS'16), to be held in Edinburgh (UK), September 8–10, 2016.

8.1.2.2. Member of the conference program committees

Jérôme Feret was member of the Program Committee of the 2nd workshop on Verification of Engineered Molecular Devices and Programs (VEMDP'15), of the Program Committee of the 7th International Conference on Bioinformatics, Biocomputational Systems and Biotechnologies (BIOTECHNO'15), of the Program Committee of the 6th International Workshop on Static Analysis and Systems Biology (SASB'15), of the Program Committee of the 25th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'15), and of the Program Committee of the 11th International Workshop on Developments in Computational Models (DCM'15). Jérôme Feret is a member of the Program Committee of the 8th International Conference on Bioinformatics, Biocomputational Systems and Biotechnologies (BIOTECHNO'16), of the Program Committee of the 23rd Static Analysis Symposium (SAS'16), and of the Program Committee of the 14th International Conference on Computational Methods in Systems Biology (CMSB'16).

Xavier Rival was member of the Extended Review Committee (ERC) of the 41st ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2015 PC), and of the Program Committee of the 13th Asian Programming Languages And Systems (APLAS'15), Pohang, Korea, November 30–December 2, 2015. Xavier Rival is a member of the Program Committee of the 1st International Conference on Functional Structures and Computational Deduction (FSCD'16), to be held in Porto, Portugal, June 2016, and of the Program Committee of the European Symposium On Programming (ESOP'17), 2017.

Cezara Drăgoi was a member of 15th International Conference on Application of Concurrency to System Design (ACSD'15), 41st International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'15), and 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNACS'15).

Vincent Danos was a member of the Program Committee of workshop on Integrative Cell Models, Lorentz Workshop, Leiden, and of the 7th Conference on Reversible Computation (RC'15), Grenoble.

8.1.2.3. Reviewer

Jérôme Feret was a reviewer for the 41st ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2015), the 18th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'15), POPL 2015, the 27th International Conference on Computer Aided Verification (CAV'15), for the 26th International Conference on Concurrency Theory (CONCUR 2015), the International Conference on Embedded Software (EMSOFT'15).

Cezara Drăgoi was a reviewer for 27th International Conference on Computer Aided Verification (CAV'15), 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'16), and 30th Annual ACM/IEEE Symposium on Logic In Computer Science (LICS'15).

8.1.3. Journal

8.1.3.1. 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.

Vincent Danos is a member of the editorial board of Mathematical Structures in Computer Science journal (MSCS) and Logical Methods in Computer Science journal (LMCS).

8.1.3.2. Reviewer - Reviewing activities

Jérôme Feret was a reviewer for Information and Computation, Theoretical Computer Science, and Science of Computer Programming.

Xavier Rival was a reviewer for ACM TOPLAS (Transactions On Programming Languages and Systems).

8.1.4. Invited talks

Jérôme Feret was invited to give a talk on “the ANASTASEC project” at a joint meeting between Inria and CISCO, on “Model reduction of differential systems” at the working group “Model Reduction of Discrete models” (GDT BIOSS), at “La demi-heure de science” (Inria Paris-Rocquencourt), and at the Seminar of the Computer Science Department of the École normale supérieure, on “An algebraic approach for inferring and using symmetries in rule-based models” at the Seminar of the team PLUME (ENS Lyon).

Jérôme Feret was invited to give a speed talk on the ÁSTRÉE analyzer at the joint workshop between École normale supérieure and SAGEM, a speed talk on rule-based modeling in the international workshop on “Integrative Cell Models” (Leiden, Netherlands) and at the joint seminar between École normale supérieure and Pasteur Institute, and a speed talk on “An algebraic approach for inferring and using symmetries in rule-based models” at the first meeting of the Working group on Symbolic Systems Biology (GDT BIOSS).

Xavier Rival was invited to give a talk on “Modular design of static analyzers for program verification” at the KAIST Colloquium (KAIST, Daejeon, Korea), on May 4th, 2015.

Xavier Rival participated to the 56th Meeting of the IFIP Working Group 2.4 in Bopphard (Germany), and gave a talk on “Non contiguous array abstraction”.

Xavier Rival was invited to give a talk on “Construction of abstract domains for heterogeneous memory properties” in October 2015.

Cezara Drăgoi gave an invited talk on “PSync: A partially synchronous language for fault-tolerant distributed algorithms" at Dagstuhl Seminar 15191, “Compositional Verification Methods for Next-Generation Concurrency", in May 2015, at LIAFA, University Paris 7 in the Verification seminar, and at the workshop on “The Chemistry of Concurrent and Distributed Programming II”", in June 2015.

Cezara Drăgoi gave an invited talk on “A Logic-based Framework for Verifying Consensus Algorithms" at Dagstuhl Seminar 15451 “Verification of Evolving Graph Structures".

Cezara Drăgoi gave a tutorial on “Static analysis for data structures" at DACS in July 2015, and an invited talk at "La demi-heure de la science" on “Logic-based static analysis for the verification of programs with dynamically allocated data structures".

Vincent Danos was invited to talk about “Computational Systems Biology” at Alan Turing Institute Scoping Workshop on networks, London, on Dec 14th, 2015.

Vincent Danos was invited to talk about “Design, optimization, and control in systems and synthetic biology" at Paris, ENS, on Nov 12th 2015.

Vincent Danos was invited to talk about “Models of growth" and the challenge of modelling processes which are embedded in a bigger adaptive system, keynote at the 16th International Workshop on Static Analysis and Systems Biology (SASB’15).

Vincent Danos was invited to talk about “Moment Semantics for Reversible Rule-Based Systems", keynote at 7th Conference on Reversible Computation (RC’15), Grenoble.

8.1.5. Leadership within the scientific community

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

8.2. Teaching - Supervision - Juries

8.2.1. Teaching

Licence:

- Jérôme Feret, and Cezara Drăgoi, Mathematics, 40h, L1, FDV Bachelor program (Frontiers in Life Sciences (FdV)), Université Paris-Descartes, France.
- Antoine Miné and Xavier Rival, Semantics and applications to verification, 48h ETD, L3, École Normale Supérieure, Paris, France.
- Xavier Rival, Introduction to static analysis, 8h, L3, École des Mines de Paris, Paris, France.
- Xavier Rival, Introduction to algorithmics, 40h ETD, L3, École Polytechnique, Palaiseau, France.

Master:

- 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.
- Xavier Rival, Introduction to abstract interpretation, 8h, M1-M2, KAIST, Daejeon, Korea.
- Xavier Rival, Experimental Course on "Program Verification", 20h ETD, M1, École Polytechnique, Palaiseau, France.
- Xavier Rival, Protocol safety and verification 20h ETD, M2 program in Avanced Communications and Network (École Polytechnique and Telecom Paris), France.
- Jérôme Feret has been invited to give a session (2h) in the Master M2IF (ÉNS Lyon) in the context of Russ Harmer's course on Rule-based modelling.

8.2.2. Supervision

PhD:

- Arlen Cox, Parametric heap abstraction for dynamic languages libraries, École Normale Supérieure, 13th of April 2015, supervised jointly by Bor-Yuh Evan Chang (University of Colorado), Xavier Rival (Inria / École Normale Supérieure) and Sriram Sankaranarayana (University of Colorado).
- Caterina Urban, Static analysis by abstract interpretation of functional temporal properties of programs, École Normale Supérieure, 29th of July 2015, supervised by Antoine Miné (CNRS / École Normale Supérieure) and Radhia Cousot (CNRS / École Normale Supérieure).
- Tie Cheng, Static analysis of spreadsheet applications, École Normale Supérieure, 23rd of September 2015, supervised by Xavier Rival (Inria / École Normale Supérieure).

PhD in progress:

- Mehdi Bouaziz, Static analysis of security properties by abstract interpretation. November 2011, Patrick Cousot and Jérôme Feret, École Normale Supérieure.

- Ferdinanda Camporesi, Abstraction of Quantitative Semantics of Rule-based models, January 2009, Radhia Cousot and Jérôme Feret (co-directed thesis with Maurizio Gabrielli, University of Bologna).
- Hugo Illous, Verification of memory properties by abstract interpretation, October 2015, Xavier Rival (co-directed thesis with Mathieu Lemerre, CEA Saclay).
- Huisong Li, Static Analysis of Data-Structures with Sharing, November 2014, Xavier Rival, École Normale Supérieure
- Jiangchao Liu, Verification of the memory safety of a micro-kernel, December 2013, Xavier Rival, École Normale Supérieure
- Antoine Toubhans, Combination of shape abstract domains, October 2011, Xavier Rival, École Doctorale de Paris Centre
- Thibault Suzanne, Thread-modular static analysis of concurrent programs in the presence of weak memory models, October 2014, Antoine Miné, École Normale Supérieure.
- Guillaume Terradot, “Modeles de croissance bacterienne/determinants de la taille", started in 2015, Vincent Danos, École Normale Supérieure.
- William Waites, “Strategie evolutionnaire dans des tissus avec interactions mecaniques" started in 2015, Vincent Danos, École Normale Supérieure.
- Andreea Beica, “Approximations max-paralleles de modeles stochastiques", started in 2015, Vincent Danos, École Normale Supérieure.

Internships:

- Eleonore Bellot did an internship at ENS with Vincent Danos on “Modeles energetiques de la cellule".
- Xavier Zaoui did an internship at ENS with Vincent Danos on “Croissance bacterienne et interferences entre antibiotiques".
- Danae David did an internship at ENS with Vincent Danos on "Representation des connaissances en biochimie des voies de signalisation".
- Ihab Boulas did an internship at ENS with Vincent Danos on “Modeles de transmission de plasmides "editeurs".

8.2.3. Juries

Jérôme Feret was reviewer of the PhD thesis of Peter Kreyßig (Friedrich-Schiller-Universität Jena, Germany, March 2015).

Jérôme Feret was examiner of the PhD thesis of Mounir Assaf (Irisa, France, May 2015).

Vincent Danos was referee on the phd thesis of Erwan Bigan, Universite Paris-Diderot, and referee on the phd thesis of Vincent Picard, Universite de Rennes 2.

8.2.4. Selection committees

Jérôme Feret was a member of the recruiting jury for junior research scientists (CR2) at Inria Paris-Rocquencourt in 2015; Jérôme Feret is a member of the recruitment committee for an assistant professor in Paris-Diderot University 2016.

9. 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

Doctoral Dissertations and Habilitation Theses

[9] C. URBAN. *Static Analysis by Abstract Interpretation of Functional Temporal Properties of Programs*, École Normale Supérieure, July 2015, https://tel.archives-ouvertes.fr/tel-01176641

Articles in International Peer-Reviewed Journals

[10] C. A. COCKRAM, M. FILATENKOVA, V. DANOS, M. EL KAROUI, D. R. F. LEACH. *Quantitative genomic analysis of RecA protein binding during DNA double-strand break repair reveals RecBCD action in vivo*, in "Proceeding of the national academy of sciences", August 2015 [*DOI :* 10.1073/PNAS.1424269112], https://hal.archives-ouvertes.fr/hal-01263631

[11] G. MISIRLI, M. CAVALIERE, W. WAITES, M. POCOCK, C. MADSEN, O. GILFELLON, R. HONORATO-ZIMMER, P. ZULIANI, V. DANOS, A. WIPAT. *Annotation of rule-based models with formal semantics to enable creation, analysis, reuse and visualisation*, in "Bioinformatics", November 2015 [*DOI :* 10.1093/BIOINFORMATICS/BTV660], https://hal.archives-ouvertes.fr/hal-01263639

Invited Conferences

[12] A. MINÉ. *AstréeA: A Static Analyzer for Large Embedded Multi-Task Software*, in "16th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'15)", Mumbai, India, Lecture Notes in Computer Science, Springer, January 2015, vol. 8931, 3 p. , https://hal.inria.fr/hal-01105235

International Conferences with Proceedings

[13] W. ABOU-JAOUDÉ, J. FERET, D. THIEFFRY. *Derivation of Qualitative Dynamical Models from Biochemical Networks*, in "CMSB 2015 : Computational Methods in Systems Biology", Nantes, France, O. ROUX, J. BOURDON (editors), Springer, September 2015, vol. 9308, pp. 195-207 [*DOI :* 10.1007/978-3-319-23401-4_17], https://hal.inria.fr/hal-01199243

[14] T. CHENG, X. RIVAL. *Static Analysis for Spreadsheet Applications for Type-Unsafe Operations Detection*, in "European Symposium On Programming (ESOP 2015)", London, United Kingdom, April 2015, https://hal.archives-ouvertes.fr/hal-01098377

[15] A. COX, B.-Y. E. CHANG, H. LI, X. RIVAL. *Abstract Domains and Solvers for Sets Reasoning*, in "20th International Conference, LPAR-20", Suva, Fiji, M. DAVIS, A. FEHNKER, A. MCIVER, A. VORONKOV (editors), November 2015, vol. Lecture Notes in Computer Science, n° 9450 [*DOI :* 10.1007/978-3-662-48899-7_25], https://hal.inria.fr/hal-01256309

[16] A. COX, B.-Y. E. CHANG, X. RIVAL. *Desynchronized Multi-State Abstractions for Open Programs in Dynamic Languages*, in "European Symposium On Programming (ESOP 2015)", London, United Kingdom, Springer, April 2015, https://hal.archives-ouvertes.fr/hal-01098379

[17] V. DANOS, T. HEINDEL, R. HONORATO-ZIMMER, S. STUCKI. *Moment Semantics for Reversible Rule-Based Systems*, in "7th International Conference on Reversible Computation", Grenoble, France, July 2015 [*DOI :* 10.1007/978-3-319-20860-2_1], https://hal.archives-ouvertes.fr/hal-01263633

[18] H. LI, X. RIVAL, B.-Y. E. CHANG. *Shape Analysis for Unstructured Sharing*, in "Static Analysis - 22nd International Symposium", Saint-Malo, France, SPRINGER (editor), September 2015, 19 p. [*DOI :* 10.1007/978-3-662-48288-9_6], https://hal.inria.fr/hal-01249418

[19] J. LIU, X. RIVAL. *Abstraction of Arrays Based on Non Contiguous Partitions*, in "15th Conference on Verification, Model Checking, and Abstract Interpretation", Mumbai, India, Springer, January 2015, n° 8931, pp. 282 - 299 [*DOI :* 10.1007/978-3-662-46081-8_16], https://hal.archives-ouvertes.fr/hal-01095985

[20] J. LIU, X. RIVAL. *Abstraction of Optional Numerical Values*, in "Programming Languages and Systems - 13th Asian Symposium, APLAS", Pohang, South Korea, X. FENG, S. PARK (editors), Springer, November 2015, vol. Lecture Notes in Computer Science, n° 9458 [*DOI :* 10.1007/978-3-319-26529-2_9], https://hal.inria.fr/hal-01256116

[21] C. URBAN, A. MINÉ. *Proving Guarantee and Recurrence Temporal Properties by Abstract Interpretation*, in "16th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'15)", Mumbai, India, Lecture Notes in Computer Science, Springer, January 2015, vol. 8931, 19 p. [*DOI :* 10.1007/978-3-662-46081-8_11], https://hal.inria.fr/hal-01105238

[22] C. URBAN. *FuncTion: An Abstract Domain Functor for Termination*, in "21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems", Londres, United Kingdom, Lecture Notes in Computer Science, Springer, April 2015, 3 p. , https://hal.inria.fr/hal-01107419

Conferences without Proceedings

[23] V. DANOS, I. GARNIER. *Dirichlet is Natural*, in "MFPS 31", Nijmegen, Netherlands, June 2015 [*DOI :* 10.1016/J.ENTCS.2015.12.010], https://hal.archives-ouvertes.fr/hal-01256903

[24] C. DRĂGOI, T. HENZINGER, D. ZUFFEREY. *The Need for Language Support for Fault-tolerant Distributed Systems*, in "1st Summit on Advances in Programming Languages (SNAPL 2015)", Dagstuhl, Germany, Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, May 2015, pp. 90–102, http://snapl.org/2015/org.html [*DOI :* 10.4230/LIPICS.SNAPL.2015.90], https://hal.inria.fr/hal-01251194

[25] 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, ACM, January 2016 [*DOI :* 10.1145/2837614.2837650], https://hal.inria.fr/hal-01251199

Books or Proceedings Editing

[26] J. FERET, A. LEVCHENKO (editors). *Proceedings of the 3rd International Workshop on Static Analysis and Systems Biology (SASB 2012)*, Electronic Notes in Theoretical Computer Science, Elsevier, Deauville, France, May 2015, vol. 313, 78 p. , https://hal.inria.fr/hal-01147670

Other Publications

[27] R. MONAT. *Thread-Modular Static Analysis by Abstract Interpretation: designing relational abstractions of interferences*, Stage effectué dans l'équipe ANTIQUE, ENS Ulm, dans le cadre d'une L3 informatique fondamentale à l'ENS Lyon, September 2015, https://hal.inria.fr/hal-01187538

[28] Y.-F. TSAI, D. COUGHLIN, B.-Y. E. CHANG, X. RIVAL. *Synthesizing Short-Circuiting Validation of Data Structure Invariants*, 2015, working paper or preprint, https://hal.inria.fr/hal-01256324

References in notes

[29] C. CHARLOTTE, M. E. KAROUI, V. DANOS, M. FILATENKOVA, D. LEACH. *Quantitative Genomic Analysis of RecA Protein Binding During DNA Double-Strand Break Repair Reveals RecBCD Action in vivo*, in "PNAS", August 2015, vol. 112, n^{o} 34, http://dx.doi.org/10.1073/pnas.1424269112

[30] 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

[31] 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

[32] V. DANOS, I. GARNIER. *Dirichlet is natural*, in "ENTCS", June 2015, to appear

[33] V. DANOS, R. HARMER, R. HONORATO-ZIMMER. *Thermodynamic Graph Rewriting*, in "Logical Methods in Computer Science", 2015, CONCUR special issue in LMCS

[34] V. DANOS, T. HEINDEL, R. HONORATO-ZIMMER, S. STUCKI. *Moment Semantics for Reversible Rule-Based Systems*, in "Proceedings of RC'15", 2015, vol. 9138

[35] G. MISIRLI, M. CAVALIERE, W. WAITES, M. POCOCK, C. MADSEN, O. GILFELLON, R. HONORATO-ZIMMER, P. ZULIANI, V. DANOS, A. WIPAT. *Annotation of rule-based models with formal semantics to enable creation, analysis, reuse and visualisation*, in "Bioinformatics", 2015, Published 11 November 2015, http://dx.doi.org/10.1093/bioinformatics/btv660

[36] A. WEISSE, D. OYARZUN, V. DANOS, P. SWAIN. *A mechanistic link between cellular trade-offs, gene expression and growth*, in "PNAS", February 2015, vol. 112, n^{o} 9, http://www.pnas.org/content/112/9/E1038.full

[37] J. WILSON-KANAMORI, V. DANOS, T. THOMSON, R. HONORATO-ZIMMER. *Kappa Rule-Based Modelling in Synthetic Biology*, in "Computational Methods in Synthetic Biology", M. A. MARCHISIO (editor), Methods in Molecular Biology, Springer, 2015, vol. 1244, pp. 105–135