RESEARCH CENTRE

Inria Lyon Centre

IN PARTNERSHIP WITH:

Université Claude Bernard (Lyon 1), Ecole normale supérieure de Lyon, CNRS

2024 ACTIVITY REPORT

Project-Team CASH

Compilation and Analyses for Software and Hardware

IN COLLABORATION WITH: Laboratoire de l'Informatique du Parallélisme (LIP)

DOMAIN

Algorithmics, Programming, Software and Architecture

THEME

Architecture, Languages and Compilation

Inria

Contents

Project-Team CASH

Creation of the Project-Team: 2019 June 01

Keywords

Computer sciences and digital sciences

A2.1. – Programming Languages

A2.1.1. – Semantics of programming languages

A2.1.2. – Imperative programming

A2.1.4. – Functional programming

A2.1.6. – Concurrent programming

A2.1.7. – Distributed programming

A2.1.10. – Domain-specific languages

A2.1.11. – Proof languages

A2.2. – Compilation

A2.2.1. – Static analysis

A2.2.2. – Memory models

A2.2.3. – Memory management

A2.2.4. – Parallel architectures

A2.2.5. – Run-time systems

A2.2.6. – GPGPU, FPGA...

A2.2.8. – Code generation

A2.3.1. – Embedded systems

A2.4. – Formal method for verification, reliability, certification

A2.4.1. – Analysis

A2.4.2. – Model-checking

A2.4.3. – Proofs

A2.5.3. – Empirical Software Engineering

A2.5.4. – Software Maintenance & Evolution

A7.2. – Logic in Computer Science

A7.2.1. – Decision procedures

A7.2.3. – Interactive Theorem Proving

A7.2.4. – Mechanized Formalization of Mathematics

A8.3. – Geometry, Topology

A8.4. – Computer Algebra

Other research topics and application domains

B3.1. – Sustainable development

B5.4. – Microelectronics

B9.5.1. – Computer science

B9.5.2. – Mathematics

1 Team members, visitors, external collaborators

Research Scientists

- Christophe Alias [INRIA, Researcher, HDR]
- Cyril Cohen [INRIA, Researcher]
- Ludovic Henrio [CNRS, Researcher, HDR]
- Gabriel Radanne [INRIA, ISFP]
- Yannick Zakowski [INRIA, Researcher]

Faculty Member

- Matthieu Moy [Team leader, UNIV LYON I, Associate Professor]

Post-Doctoral Fellow

- Emmanuel Arrighi [ENS DE LYON]

PhD Students

- Thaïs Baudon [ENS de Lyon, from Sep 2024 until Oct 2024]
- Thaïs Baudon [ENS DE LYON, until Aug 2024]
- Peio Borthelle [INRIA, from Oct 2024]
- Nicolas Chappe [INRIA, from Sep 2024 until Nov 2024]
- Nicolas Chappe [ENS DE LYON, until Aug 2024]
- Emma Nardino [ENS DE LYON, from Sep 2024]
- Oussama Oulkaid [ANIAH, CIFRE]
- Alec Sadler [INRIA]
- Hugo Thievenaz [INRIA]

Interns and Apprentices

- Melvyn Bauvent [INRIA, Intern, from Apr 2024 until Jun 2024]
- Wenke Du [INRIA, Intern, from Mar 2024 until Sep 2024]
- Léon Frenot [ENS DE LYON, Intern, from Feb 2024 until Jul 2024]
- Vivien Gachet [ENS DE LYON, Intern, from Feb 2024 until Jul 2024]
- Theo Gaige [INSA LYON, Intern, from May 2024 until Jul 2024]
- Emma Nardino [ENS DE LYON, Intern, from Apr 2024 until Jul 2024]

Administrative Assistant

- Elise Denoyelle [Inria]

External Collaborator

- Laure Gonnord [GRENOBLE INP]

2 Overall objectives

The overall objective of the CASH team is to design and develop ways to improve the quality of software. We work both on tools that help programmers write better programs, and compilers that turn these programs into efficient executables.

By *improving the quality*, we mean both the safety of programs and the efficiency of their execution. We notably provide programmers with better *programming language* constructs to express their intent: constructions that give them guarantees by-construction such as memory-safety, determinism, etc. When guarantees can't be obtained by construction, we also develop *static analyses* to detect bugs or to prove preconditions needed to apply program transformations. We use such guarantees to develop new *optimizations* to generate efficient code. All these contributions find their foundation and justification in the *semantics* of programs. Additinally, we provide *simulators* to execute programs that require a specific execution platform.

When it comes to high level programming constructs for parallelism, we develop a specific expertise in asynchronous computations and ruling out race-conditions in concurrent programs. In this realm, we propose new paradigms, but also contribute to the semantics of such programs. For instance, we design ways to specify the semantics using monadic interpreters, and use them to study the correctness of compilers.

We ensure safety guarantees both through type-systems and analyses, in vastly different contexts: from the verification of electrical circuits to the design of a new module systems for OCaml. As is recurrent in our work, we pragmatically adapt the approach to the practical application at hand.

We design code transformation for the efficient execution of programs, in particular targetting HPC. Our contributions in this realm extend the polyhedral model to make it applicable to a wider range of programs, and to bring its potential for optimisation to new kind of applications (parametric tiling, sparse structures, ...). We also design optimisations for structured data such as trees, or more generally algebraic data types.

Our Approach and methodology. We target a balance between theory and practice: problems extracted from industrial requirements often yield theoretical problems.

On the practical side, the CASH team targets applied research, in the sense that most research topics are driven by actual needs, discussed either through industrial partnership or extracted from available benchmarks.

The theoretical aspects ensure the coherency and the correctness of our approach, they are mostly based on formally defined semantics. The formalization of the different representations of the programs and of the analyses allow us to show that these different tasks are performed with the same understanding of the program semantics. Formalization is done either with pen-and-paper definitions and proofs, or through mechanized proofs using the Rocq prover[1].

Our approach is to cross-fertilize between several communities. For example, the abstract interpretation community provides a sound theoretical framework and very powerful analysis, but these are rarely applied in the context of optimizing compilation. Similarly, the formal verification community is very active on software, and RTL hardware verification, but very few researchers applied it to transistor-level verification (lower-level than RTL).

Many of our contributions are "meta-contributions", in the sense that we do not only provide final results, but also tools to help on the foundational aspects of these results. We provide theoretical and practical tools that are both used internally in the team to achieve our final results, and provided to researchers outside the team. For example, we provide tools to express and manipulate program semantics formally in novel ways instead of focusing only on certified compilers themselves. Also, our contributions to the polyhedral model not only propose new optimizations based on the model, but also

[1] "The Rocq prover" is the new name of "the Coq proof assistant", starting from end of 2024. In this document we will refer to Coq as "The Rocq prover", except in paper names or conference titles which were not (yet) updated at the time of the writing.

general purpose tools to help other researchers to prototype new optimizations. This makes it a bit easier to target general purpose conferences where meta-results are slightly more valued provided they are proven to be applicable but makes our development efforts spread out into several unrelated projects.

3 Research program

The structure of the team is illustrated by Fig. 1 with 4 research directions, detailed below. Vertical shapes represent more fundamental aspects, while horizontal ones represent the applications. Note that three of these directions are an evolution of the first three research focuses in our past activities. The team is recognized and contributes mostly in the four identified axes. Obviously, every direction potentially intersects all the others, but we emphasise the intersections between the fundamental aspects and the applications. For example, in the work on static strong typing programming language constructs are designed to provide analyses with strong guarantees. The purpose of the direction semantics and proofs is to provide tools to certify correctness of optimizations and analyses. This complementary is manifest in the upcoming ANR "Shannon Meet Cray" (sxc.inria.fr) to be started in 2025 aiming to design new computational models and languages for efficient data-processing that will require cooperation between language design, optimisations and analysis.

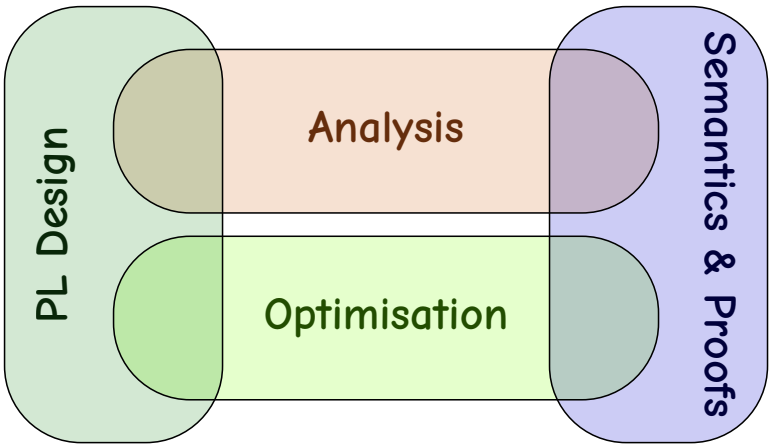

Figure 1: The CASH Research Topics

More importantly than intersection between axes, we highlight that many of the contributions we envision are shared between members of the team, and/or with external collaborators.

3.1 Programming Language Design

While parallelism is one major field of research concerning programming languages, we still have a significant research agenda in sequential programming languages. The interplay between programming languages and static guarantees is still a major concern that drives most of our language designs, in particular our expertise on type systems drives many of our design choices as it is a very effective way to ensure static properties of programs.

Participants Cyril Cohen, Ludovic Henrio, Matthieu Moy, Gabriel Radanne.

3.2 Semantics and Proofs

The team is, on overall, interested in formalization and proofs. In this research direction, we push the degree of formalization as far as it can be with machine-checked proofs, typically using the Rock prover (the new name of the "Coq proof assistant"). The following challenges in the organization of formal proofs arise: defining or reusing hierarchies of mathematical structures (adjunctions, monads, (co-)algebras,

etc), transferring proofs between structures, or turning abstract algorithms in executable and efficient code.

Participants Cyril Cohen, Ludovic Henrio, Gabriel Radanne.

3.3 Program Analysis and Verification

Our activities on program analysis serve two purposes: large scale verification, intended to find bugs in programs and circuits and help fixing them; and analysis to improve optimizing compilers (e.g. infer invariants that can be used by the optimizer) and language tools (e.g. use the type system to improve the tooling available for developers).

Participants Christophe Alias, Ludovic Henrio, Matthieu Moy, Gabriel Radanne.

3.4 Optimizations and Program Transformations

We have a special focus on *data transformations* both in the context of functional compilation and imperative compilation; while ensuring the *scalability* of our optimizations. These are *transverse* topics that we address through the following several research directions, like memory representation of datatypes, automatic or semi-automatic parallelization, etc.

Participants Christophe Alias, Cyril Cohen, Ludovic Henrio, Gabriel Radanne.

4 Application domains

The scope of targeted applications is wide, from specialized HPC applications on supercomputers to general purpose computing, from massively parallel applications to sequential code, from functional languages to imperative code, etc. Our main focus is software, but we also consider hardware, both as an execution platform for software, and as a research topic (hardware generation and hardware circuit analysis). What links all our activities is that our object of study is *computer programs*.

While a global approach links CASH activities and members, we do not have a single unified toolchain where all contributions would be implemented. Instead we try to attach ourself to existing frameworks. This implies that different activities of CASH target different application domains and potential end-users: OCaml ecosystem; Vellvm project for certified compilation; standalone prototype for memory layout for algebraic data types but this may evolve to a Rust implementation. In other words, we chose for each of our contribution which toolchain and/or application domain is the most relevant for the contribution. This allows us to both use the best technical framework and to pick the best way to eventually reach our final users.

5 Social and environmental responsibility

Footprint of research activities

Although we do not have a precise measure of our carbon (and other environmental) footprint, the two main sources of impact of computer-science research activities are usually travel (plane) and digital equipment (lifecycle of computers and other electronic devices).

Many members of the CASH team are already in an approach of reducing their international travel, and hopefully the new solutions we had to set up to continue our activities during the COVID crisis will allow us to continue our research with a sustainable amount of travel, and using other forms of remote collaborations when possible.

As far as digital equipment is concerned, we try to extend the lifetime of our machines as much as possible.

Impact of research results

Many aspects of our research are meant to provide tools to make better programs, in particular avoid bugs and improve power-efficiency. It is very hard, however, to assess the actual impact of such research. For example, our work on avoiding bugs may either avoid waste of resource due to buggy software or help developers write code for larger, resource-consuming execution platforms. In many cases, improvements in power-efficiency lead to a rebound effect which may weaken the benefit of the improvement, or even lead to an increase in total consumption (backfire).

CASH provides tools for developers, but does not develop end-user applications. We believe the social impact of our research depends more on the way developers will use our tools than on the way we conduct our research.

Ludovic Henrio followed the “Atelier Sciences Environnements Sociétés Inria 2021” (atelier Sens) organized by Eric Tannier in June 2021. Then, for the voluntary Cash members, he has animated an atelier Sens during the Cash seminar in October 2021.

One line of research explicitly targets energy and environmental transition. We study the possibility of a degrowth in computing power, trying to tackle the issue both from the technical point of view (e.g. our memory-consumption measurements as part of the Fruganum project) and human sciences point of view.

6 New software, platforms, open data

6.1 New software

6.1.1 DCC

Name: DPN C Compiler

Keywords: Polyhedral compilation, Automatic parallelization, High-level synthesis

Functional Description: Dcc (Data-aware process network C Compiler) compiles a regular C kernel to a data-aware process network (DPN), a dataflow intermediate representation suitable for high-level synthesis in the context of high-performance computing. Dcc has been registered at the APP ("Agence de protection des programmes") and transferred to the XtremLogic start-up under an Inria license.

News of the Year: This year, Dcc was enhanced with user-guided loop tiling. Given a user-specified tiling template, a correct loop tiling with minimal latency is inferred.

Publication: hal-03143777

Contact: Christophe Alias

Participants: Christophe Alias, Alexandru Plesco

6.1.2 PoCo

Name: Polyhedral Compilation Library

Keywords: Polyhedral compilation, Automatic parallelization

Functional Description: PoCo (Polyhedral Compilation Library) is framework to develop program analysis and optimizations in the polyhedral model. PoCo features polyhedral building blocks as well as state-of-the-art polyhedral program analysis. PoCo has been registered at the APP (“agence de protection des programmes”) and transferred to the XtremLogic start-up under an Inria licence.

News of the Year: This year, GLPK was interfaced to the symbolic engine. Also, the Farkas engine was improved to handle more complex affine constraints.

Contact: Christophe Alias

Participant: Christophe Alias

6.1.3 fkcc

Name: The Farkas Calculator

Keywords: DSL, Farkas Lemma, Polyhedral compilation

Scientific Description: fkcc is a scripting tool to prototype program analyses and transformations exploiting the affine form of Farkas lemma. Our language is general enough to prototype in a few lines sophisticated termination and scheduling algorithms. The tool is freely available and may be tried online via a web interface. We believe that fkcc is the missing chain to accelerate the development of program analyses and transformations exploiting the affine form of Farkas lemma.

Functional Description: fkcc is a scripting tool to prototype program analyses and transformations exploiting the affine form of Farkas lemma. Our language is general enough to prototype in a few lines sophisticated termination and scheduling algorithms. The tool is freely available and may be tried online via a web interface. We believe that fkcc is the missing chain to accelerate the development of program analyses and transformations exploiting the affine form of Farkas lemma.

Release Contributions: - Script language - Polyhedral constructors - Farkas summation solver

URL: http://foobar.ens-lyon.fr/fkcc/

Publication: hal-03106000

Contact: Christophe Alias

Participant: Christophe Alias

6.1.4 Vellvm

Keywords: Coq, Semantic, Compilation, Proof assistant, Proof

Scientific Description: A modern formalization in the Coq proof assistant of the sequential fragment of LLVM IR. The semantics, based on the Interaction Trees library, presents several rare properties for mechanized development of this scale: it is compositional, modular, and extracts to a certified executable interpreter. A rich equational theory of the language is provided, and several verified tools based on this semantics are in development.

Functional Description: Formalization in the Coq proof assistant of a subset of the LLVM compilation infrastructure.

URL: https://github.com/vellvm/vellvm

Contact: Yannick Zakowski

Participants: Yannick Zakowski, Steve Zdancewic, Calvin Beck, Irene Yoon

Partner: University of Pennsylvania

6.1.5 ribbit

Keywords: Compilation, Pattern matching, Algebraic Data Types

Functional Description: Ribbit is a compiler for pattern languages with algebraic data types which is parameterized by the memory representation of types. Given a memory representation, it generates efficient and correct code for pattern matching clauses.

URL: https://gitlab.inria.fr/cash/ribbit

Contact: Gabriel Radanne

6.1.6 adtr

Name: ADT Rewriting language

Keywords: Compilation, Static typing, Algebraic Data Types, Term Rewriting Systems

Functional Description: ADTs are generally represented by nested pointers, for each constructors of the algebraic data type. Furthermore, they are generally manipulated persistently, by allocating new constructors.

ADTr allow representing ADTs in a flat way while compiling a pattern match-like construction as a rewrite on the memory representation. The goal is to then use this representation to optimize the rewriting and exploit parallelism.

URL: https://github.com/Drup/adtr

Publication: hal-03355377

Contact: Gabriel Radanne

Participants: Gabriel Radanne, Paul Iannetta, Laure Gonnord

6.1.7 dowsing

Keywords: Static typing, Ocaml

Functional Description: Dowsing is a tool to search function by types. Given a simple OCaml type, it will quickly find all functions whose types are compatible.

Dowsing works by building a database containing all the specified libraries. New libraries can be added to the database. It then builds an index which allow to quickly answer to requests.

URL: https://github.com/Drup/dowsing/

Publication: hal-03355381

Contact: Gabriel Radanne

Participants: Gabriel Radanne, Laure Gonnord

6.1.8 odoc

Keyword: Ocaml

Functional Description: OCaml is a statically typed programming language with wide-spread use in both academia and industry. Odoc is a tool to generate documentation of OCaml libraries, either as HTML websites for online distribution or to create PDF manuals and man pages.

URL: https://github.com/ocaml/odoc/

Contact: Gabriel Radanne

Participants: Jon Ludlam, Gabriel Radanne, Florian Angeletti, Leo White

6.1.9 PoLA

Name: PoLA: a Polyhedral Liveness Analyser

Keywords: Polyhedral compilation, Array contraction

Functional Description: PoLA is a C++ tool that optimizes the footprint of C(++) programs of the polyhedral model by applying reduced mappings deduced from dynamic analysis of the program. More precisely, we apply a dataflow analysis on traces of a program, obtained either by execution or interpretation, and infer parametrized mappings for the arrays used for intermediate computations. This tool is part of the Polytrace project.

URL: https://hthieven.gitlabpages.inria.fr/pola/

Publications: thievenaz:hal-03862219, thievenaz:hal-03862218

Contact: Christophe Alias

Participants: Hugo Thievenaz, Christophe Alias, Keiji Kimura

Partner: Waseda University

6.1.10 Actors-OCaml

Keywords: Concurrency, Ocaml

Functional Description: An actor library for OCaml

URL: https://gitlab.inria.fr/mandrieu/actors-ocaml

Contact: Gabriel Radanne

6.1.11 ctrees

Name: Choice Trees

Keywords: Coq, Concurrency, Formalisation, Semantics, Proof assistant

Functional Description: We develop so-called "ctrees", a data-structure in Coq suitable for modelling and reasoning about non-deterministic programming languages as an executable monadic interpreter. We link this new library to the Interaction Trees project: ctrees offer a valid target for interpretation of non-deterministic events.

URL: https://github.com/vellvm/ctrees/

Contact: Yannick Zakowski

6.1.12 ERCtool

Name: Electrical Rule Checking Tool

Keywords: Verification, Model Checking, Electrical circuit, Transistor, Program verification, Formal methods

Functional Description: ERCtool is developed as part of a collaboration with the Aniah company, specialized in the verification of electric properties on circuits. A specificity of ERCtool is that it allows the analysis of a circuit with multiple power-supplies, and allows getting formal guarantees on the absence of error.

Contact: Matthieu Moy

Partner: Aniah

6.1.13 Math-Components

Name: Mathematical Components library

Keywords: Proof assistant, Coq, Formalisation

Functional Description: The Mathematical Components library is a set of Coq libraries that cover the prerequisites for the mechanization of the proof of the Odd Order Theorem.

URL: https://math-comp.github.io/

Contact: Assia Mahboubi

Participants: Alexey Solovyev, Andrea Asperti, Assia Mahboubi, Cyril Cohen, Enrico Tassi, Francois Garillot, Georges Gonthier, Ioana Pasca, Jeremy Avigad, Laurence Rideau, Laurent Théry, Russell O'Connor, Sidi Ould Biha, Stéphane Le Roux, Yves Bertot

6.1.14 Math-comp-analysis

Name: Mathematical Components Analysis

Keywords: Proof assistant, Coq, Formalisation

Functional Description: This library adds definitions and theorems to the Math-components library for real numbers and their mathematical structures.

Release Contributions: First major release, several results in topology, integration, and measure theory have been added.

URL: https://github.com/math-comp/analysis

Publications: hal-02463336, hal-03917948, hal-01719918

Contact: Cyril Cohen

Participants: Cyril Cohen, Georges Gonthier, Marie Kerjean, Assia Mahboubi, Damien Rouhling, Pierre Roux, Laurence Rideau, Pierre-Yves Strub, Reynald Affeldt, Laurent Théry, Yves Bertot, Zachary Stone

Partners: Ecole Polytechnique, AIST Tsukuba, Onera

6.1.15 Hierarchy Builder

Keywords: Coq, Metaprogramming

Scientific Description: It is nowadays customary to organize libraries of machine checked proofs around hierarchies of algebraic structures. One influential example is the Mathematical Components library on top of which the long and intricate proof of the Odd Order Theorem could be fully formalized. Still, building algebraic hierarchies in a proof assistant such as Coq requires a lot of manual labor and often a deep expertise in the internals of the prover. Moreover, according to our experience, making a hierarchy evolve without causing breakage in client code is equally tricky: even a simple refactoring such as splitting a structure into two simpler ones is hard to get right. Hierarchy Builder is a high level language to build hierarchies of algebraic structures and to make these hierarchies evolve without breaking user code. The key concepts are the ones of factory, builder and abbreviation that let the hierarchy developer describe an actual interface for their library. Behind that interface the developer can provide appropriate code to ensure retro compatibility. We implement the Hierarchy Builder language in the hierarchy-builder addon for the Coq system using the Elpi extension language.

Functional Description: Hierarchy Builder is a high level language for Coq to build hierarchies of algebraic structures and to make these hierarchies evolve without breaking user code. The key concepts are the ones of factory, builder and abbreviation that let the hierarchy developer describe an actual interface for their library. Behind that interface the developer can provide appropriate code to ensure retro compatibility.

Release Contributions: Compatible with Coq 8.18 to Coq 8.20. Added support for instance saturation

URL: https://github.com/math-comp/hierarchy-builder

Publication: hal-02478907

Contact: Enrico Tassi

Participants: Enrico Tassi, Cyril Cohen

Partners: University of Tsukuba, Onera

6.1.16 Trocq

Keywords: Proof synthesis, Proof transfer, Coq, Elpi, Logic programming, Parametricity, Univalence

Functional Description: Trocq is a prototype of a modular parametricity plugin for Coq, aiming to perform proof transfer by translating the goal into an associated goal featuring the target data structures as well as a rich parametricity witness from which a function justifying the goal substitution can be extracted.

The plugin features a hierarchy of parametricity witness types, ranging from structure-less relations to a new formulation of type equivalence, gathering several pre-existing parametricity translations, including univalent parametricity and CoqEAL, in the same framework.

This modular translation performs a fine-grained analysis and generates witnesses that are rich enough to preprocess the goal yet are not always a full-blown type equivalence, allowing to perform proof transfer with the power of univalent parametricity, but trying not to pull in the univalence axiom in cases where it is not required.

The translation is implemented in Coq-Elpi and features transparent and readable code with respect to a sequent-style theoretical presentation.

Release Contributions: Support for the Prop sort

URL: https://github.com/coq-community/trocq

Publication: hal-04177913

Contact: Cyril Cohen

Participants: Cyril Cohen, Enzo Crance, Assia Mahboubi

Partner: Mitsubishi Electric R&D Centre Europe, France

7 New results

This section presents the scientific results obtained in the evaluation period. They are grouped according to the directions of our research program.

7.1 Research direction 1: Programming Language Design

7.1.1 Actors, futures, and algebraic effects

Participants: Vivien Gachet, Wenke Du, Ludovic Henrio, Gabriel Radanne.

This work aims to provide high level language constructors for concurrency and parallelism like actors and futures using modern functional language constructs like algebraic effects. Actors have been implemented and successfully used in several industry-grade frameworks, such as Akka. Algebraic effects allow the precise modelling of operation with effects, while providing excellent composition properties. They have been used both as a fundamental primitive for theoretical study, but also used as effective building blocks to create new complex control and effectful operators. The new version of OCaml with multicore support promotes the use of algebraic effects to implement new concurrency primitives. We implement actors using algebraic effects, and obtain a practical, efficient implementation of Actors for OCaml.

This year we have two new contributions in this line of work:

- We extended the Actor library with support for distributed computations enabling the use of our actor library to coordinate a computation made on several machines. This implementation of this aspect required to deal with communication of values, identification of remote actors, broadcast of replies over the network. We have a proof of concept implementation of distributed actors with

several limitations but showing that the approach is promising. We presented it in the OCaml Workshop 2024 [19]

- We investigated the notion of tail-calls in the domain of asynchronous calls (with futures). Indeed, like sequential tail calls, it is possible to optimise tail asynchronous calls, avoiding the creation of extra futures for representing the results and limiting the number of context-switch between threads. We formalised the approach, designed a first implementation of the context, and are currently working on the proof of correctness of the optimisation. We presented it in the OCaml Workshop 2024 [20]

7.1.2 Software Ecosystems for Digital Frugality

Participants: Matthieu Moy, Guillaume Salagnac, Théo Gaige, Melvyn Bauvent.

The environmental impact of digital equipment is a concern of growing importance. The carbon footprint of the domain is estimated to 2-4% and most scenario for the next decades anticipate a growth. Manufacturing digital devices requires a wide range of rare elements, that become rarer and rarer as we extract them and that are very hard to recycle. The current trajectory of the digital domain is not sustainable in the long term, and we will probably need to replace the current "infinite growth" paradigm with some form of degrowth in the future. Technological progress on efficiency may be part of the solution, but they are usually cancelled by rebound effect. We claim that a reflection on a potential reduction of the overall computational power is needed. If not well anticipated, a degrowth may be imposed to us rather than chosen. We started working on anticipating such digital degrowth scenarios.

We currently tackle the question of digital ecosystems (programming language, build tool chain, execution environments, editing tools) for frugal systems. Assuming the computational power of computers is reduced in the future, some tools stop being usable by lack of resources. We consider that the main bottleneck would be physical RAM. We developed a tool called mprobe that measures the memory consumption of a program or a set of programs, and a set of scripts to launch this tool on a variety of programs, written in different programming language. We measure both the resource needed to execute the programs, to build it (compile, link, etc.), and also the resource needed to build the tool chain itself. We are working on the analysis of the results. Preliminary results show that even "old" languages like C and C++ require relatively large amounts of RAM to build simple programs, and that interpreted languages like Python may be viable candidates, although their resource consumption in terms of CPU is higher.

7.1.3 A Language of Patterns for Control Flow Graphs

Participants: Léon Frénot, Gabriel Radanne, Yannick Zakowski.

During his M2 internship, Léon has worked on the design in the Rocq proof assistant of a language of patterns to decompose an open control flow graph. Consider for instance a block fusion optimization: a pattern allows for decomposing the graph of interest into two blocks whose fusion is valid, and the remaining of the block. The language of pattern comes with a collecting semantics, and a propositional specification.

This first experimentation has led to a more elegant and general proof of block fusion that the one previously exsting in Vellvm. Further research is necessary to validate the value of the approach.

7.2 Research direction 2: Semantics and Proofs

7.2.1 Tooling for the Rocq prover and entanglement with category theory

Participants: Cyril Cohen, Quentin Vermande, Reynald Affeldt *(AIST, Japan)*, Xavier Allamigeon *(Ecole polytechnique, Paris)*, Samuel Arsac *(PLUME, LIP, ENS de Lyon)*, Russel Harmer *(PLUME, LIP, ENS de Lyon)*, Marie Kerjean *(LIPN, Paris)*, Noah Loutchmia *(Gallinette, Inria, Nantes)*, Damien Pous *(PLUME, LIP, ENS de Lyon)*, Pierre Roux *(Onera, Toulouse)*, Kazuhiko Sakaguchi *(Galinnette, Inria then LIP, ENS de Lyon)*, Vojtěch Štěpančík *(Gallinette, Inria, Nantes)*, Zachary Stone, Enrico Tassi *(STAMP, Inria, Sophia Antipolis)*, Paolo Torrini *(STAMP, Inria, Sophia Antipolis)*.

This axis deals with the contribution to the Rocq prover libraries to either automate the structuring (using Hierarchy Builder) and transfer (using Trocq) of properties, or provide general mathematical results (using the mathematical components library).

Trocq is both a new calculus performing a variant of a parametricity translation to achieve transfer of theorems for the calculus of constructions [16], and a prototype plugin for the Rocq prover.

Hierarchy Builder (HB) is a domain specific language integrated to Rocq and designed to formally describe hierarchies of mathematical structures (e.g. Groups, Rings, Vector Spaces, etc), their various equivalent axiomatisations, their generic theory and their instances. As the impact of this project grows, several axes have emerged. In the context of the CoREACT ANR, we apply and extend HB to provide concise formalization of categories, adhesive categories, internal categories and double categories. We also extend the Math-Components and math-comp-analysis libraries with several structures (including preorders, semi-modules, and topological vector spaces). This is a stress test for HB and many contributors – including Quentin Vermande as part of his PhD thesis – improved and optimized HB or the Rocq prover.

As part of Vojtěch Štěpančík's PhD, we also worked towards the automated reconstruction of structure from partial descriptions, e.g.: reconstructing the definition of morphisms from the description of objects, the morphism part of a functor from the object part, and naturality properties.

In the current state of the Mathematical Components library, finite sets (represented as finite predicates over some type) are only allowed on finite types (types for which there exists an enumeration function of their elements). We are trying to lift this limitation without breaking compatibility with projects depending on it (more than 500000 lines of publicly known code).

7.2.2 Formalisation of Cylindrical Algebraic Decomposition (CAD) in the Rocq prover

Participants: Quentin Vermande.

As part of Vojtěch Štěpančík's PhD, we wrote and formally certified Collins' CAD algorithm in Rocq. This is one more step towards the efficient and certified implementation of quantifier elimination for the theory of real closed fields. In Section 7.4 we explain the potential future impact on some rigorous optimization steps for compilers.

7.2.3 Operational Game Semantics

Participants: Peio Borthelle, Tom Hirschowitz, Guilhem Jaber, Yannick Zakowski.

Peio Borthelle has been dedicating his PhD to the formalization in Rocq of the Operational Game Semantics (OGS) approach. OGS is a technique used to construct sound models w.r.t. contextual equivalence for higher-order languages, in which names are exchange in lieu of higher order values.

This year has seen a great conclusion to this project: - Peio's work [23] has been accepted for publication at ESOP'25. - Peio has written his PhD manuscript, currently under revision from Stephanie Weirich and Damien Pous. He will defend on March the 12th, 2025.

7.2.4 Deterministic parallel programs

Participants: Ludovic Henrio, Yannick Zakowski, Emma Nardino, Violet Ka I Pun, Einar Broch Johnsen, Asmund Kløvstad.

This research direction is mainly driven by visits and remote meetings between Ludovic Henrio, Yannick Zakowski and our Norwegian colleagues. First results were published in 2021 on a simple static criteria for deterministic behaviour of active objects. We are now extending this work to be able to ensure deterministic behaviour in more cases and to lay a theoretical background that will make our results more general and easier to adapt to different settings.

Last year, we have formalised in the Rocq prover a result by DeBruijn dating back from the 70th on proving confluence of a system. In the process, we have solved some mistakes in the existing proof, and generalised it in a way that will make it even more useful in the context of programming language semantics. By now, the application to a first simple confluence condition where many threads interact with a single stateless server is fully proved correct. We are now extending the result to a sortt of interacting stateless servers. The proof has revealed to be more difficult than expected but we project wrapping it up and submitting a paper during the first trimester of 2025.

Orthogonally, Emma Nardino has extended this year the previous results with a more flexible confluence property (enabling proof of confluence modulo equivalence relation). She has also investigated the extension of the language to futures.

7.2.5 A formal, compositional, modular and executable semantics for LLVM IR

Participants: Calvin Beck, Hanxi Chen, Irene Yoon, Steve Zdancewic, Yannick Zakowski.

Yannick Zakowskimaintains for several years a collaboration with the University of Pennsylvania, most notably around the formalisation in the Rocq prover of the semantics of LLVM IR. The project, dubbed Vellvm, aims for fidelity to the reference, but also maintainability, calling for modern semantic approaches such as Interaction Trees (itrees) [27].

A notably new restult this year is the publication at ICFP'24 [10] of the description of the new memory model, whose main purpose is to increase fidelity, and provide better support for justifying optimizations in the presence of pointer to integer casts. The core intuition behind this development is to pretend that the memory is infinite during the first swarm of optimizations.

7.2.6 Verified Compilation Infrastructure for Concurrent Programs

Participants: Nicolas Chappe, Ludovic Henrio, Yannick Zakowski.

The objective of this research direction is to provide semantic and reasoning tools for the formalization of concurrent programs and the verification of compilers for concurrent languages. In particular, we want to apply these results to the design of verified optimizing compilers for parallel high-level languages. We wish to proceed in the spirit of the approach advocated in Vellvm [28]: compositional, modular, executable monadic interpreters based on Interaction Trees [27] are used to specify the semantics of the language, in contrast with more traditional transition systems. Proving correct optimizations for such concurrent languages naturally requires new proof techniques that we need to design as well. Last year had seen the successful publication of the ctrees project. This year's major contributions in this line of work are:

- With Nicolas Chappe, we have stabilised the meta-theories of CTrees that include an alternate definition of strong bisimilarity and strong similarity enjoying better proof principles.

- Nicolas Chappe has developped a minimal concurrent version of Vellvm that formalises llvm using ctrees to reason about its semantics in presence of threads and a weak memory models. The approach is modular and suitable for various concurrent memory models.

- These new results have been packaged into two publications: an extended version of our POPL paper currently under submission at JFP, and a paper [24] accepted at CPP'25 describing the model for concurrent LLVM.

- Nicolas Chappe has written up his manuscript, and successfully defended his PhD on November 22nd!

7.2.7 A new module system for OCaml

Participants: Clement Blaudeau, Didier Remy, Gabriel Radanne.

ML modules offer large-scale notions of composition and modularity. Provided as an additional layer on top of the core language, they have proven to be both vital to the working OCaml and SML programmers, and inspiring to other use-cases and languages. Unfortunately, their meta-theory remains difficult to comprehend, requiring heavy machinery to prove their soundness.

We study a translation from ML modules to F_ω to provide a new comprehensive description of OCaml modules, embarking on a journey right from the source OCaml module system, up to F_ω , and back. This year, we published our formal description of OCaml modules via an F_ω translation [12]. We then used those insights to develop a new model directly for OCaml [11]. Clement Blaudeau defended his PhD on 11th of December 2024.

7.3 Research direction 3: Program Analysis and Verification

7.3.1 Formal Verification of Electric Properties on Transistor-Level Descriptions of Circuits

Participants: Oussama Oulkaid, Bruno Ferres, Ludovic Henrio, Matthieu Moy, Gabriel Radanne, Mehdi Khosravian.

We started discussions with the Aniah start-up in 2019, and started a formal partnership in 2022, with the recruitment of Bruno Ferres as a post-doc, and Oussama Oulkaid as a CIFRE Ph.D (co-supervised by Aniah, Verimag, and LIP). We developed a prototype verification tool. The tool compiles transistor-level circuit descriptions (CDL file format) to logical formula expressing the semantics of the circuit plus a property to verify, and uses an SMT solver (Z3) to check the validity of the property. The tool was successfully used on a real-life case study, and we showed that our approach can reduce the number of false-alarms significantly compared to traditional approaches, with a reasonable computational cost (under a second for most sub-circuits analyzed). To the best of our knowledge, formal methods like SAT/SMT-solving were never applied to multi-supplies electronic circuits before. We published these results at DATE 2024 [17]. The technique experimented in the prototype was successfully re-implemented in the production tool commercialized by Aniah and is now available in the released version.

In 2024, we developed a richer semantics able to take into account more quantitative aspects on the circuits under analysis, and applied it to circuit reliability analysis (find the worst-case configuration in terms of circuit aging) and electrical over-stress (check that the voltage difference applied to each transistor is never larger than a maximum allowed value). We are working on the publication of these results (one conference paper submitted, one journal paper to be submitted by the end of the year).

In parallel with the technical work, we conducted a thorough review of existing work on the domain, and submitted a survey article to the TODAES journal.

7.3.2 Verified Abstract Interpreters as Monadic Interpreters

Participants: Laure Gonnord, Sébastien Michelland, Yannick Zakowski.

In the realm of verified compilation, one typically wants to verify the static analyzes used by the compiler. In existing works, the analysis is typically written as a fuel-based pure function in Coq and verified against the semantics described as a transition system. The goal of this research is to develop the tools and reasoning principles to transfer these ideas to a context where the semantics of the language is defined as a monadic interpreters built on Interaction Trees.

We have managed to come back to this ambitious project started during Sébastien's internship two years ago and give it an extremely satisfying conclusion: it has been published at ICFP'24 [15].

7.3.3 Search functions by types

Participants: Gabriel Radanne, Laure Gonnord, Emmanuel Arrighi.

In this research direction, we investigate how to allow programmers to search a function by their type. The type can be either provided explicitly by programmers, or obtained directly from the editor while programming. Given this type, our goal is to search a collection of libraries and return a list of functions whose type "corresponds" to the query. We implemented Dowsing [26](6.1.7), which imlpements this idea in the OCaml ecosystem. This year, as part of the postdoc of Emmanuel Arrighi, we developed significant improvements (×500 speed) to type unification algorithms [18], allowing us to scale this search to the whole OCaml ecosystem. We are now implementing this work in existing OCaml tools, notably Sherlodoc.

7.4 Research direction 4: Optimizations and Program Transformations

7.4.1 Scalable Array Contraction with Trace Analysis

Participants: Hugo Thievenaz, Keiji Kimura, Christophe Alias.

In this work, we seek to simplify polyhedral optimizations by leveraging simple analysis on a *few off-line execution traces*. The main intuition being that, since polyhedral transformations are expressed as affine mappings, only a few points are required to infer the general mapping. We focus on array contraction, a well known technique to reallocate temporary arrays thanks to affine mappings so the array size is reduced. This year, we made the following contributions:

- We developed an algorithm to *extrapolate* array liveness analysis from an execution trace, using a dedicated *widening* operator.
- We proposed an algorithm to derive a *linear* array allocation, which improve the memory footprint by a constant factor.
- We demonstrate experimentally the *scalability* and the *accuracy* of our method on the benchmarks of the polyhedral compilation community.

These contributions are part of the PhD thesis of Hugo Thievenaz, defended this year. A journal submission is under preparation.

7.4.2 Automatic Specialization of Dense Code on Sparse Structures

Participants: Alec Sadler, Christophe Alias.

This work is concerned with the *automatic parallelization of sparse code*. Our approach is to *specialize at runtime* a canonical dense kernel on a sparse structure and to extract the runtime kernels to be distributed on the target architecture. This year, we focused on the *specialization* part. We proposed an algorithm able to propagate the sparsity across a dense computation, which may includes *reductions*. Our approach leverages an *abstraction* using *regular expressions* and particularly *Kleene iterations* to summarize the effect of loop nests. Experimental evaluation shows the precision of our approach.

These results are part of the PhD thesis of Alec Sadler and will be presented to IMPACT'25.

7.4.3 Memory optimizations for Algebraic Data Types

Participants: Thaïs Baudon, Gabriel Radanne, Laure Gonnord.

Algebraic Data Types (ADT) have emerged as an incredibly effective tool to model and manipulate data in programs. ADTs also provide numerous advantages for optimizing compilers, as the rich declarative description allows choosing the memory representation of the types. Initially found in functional programming languages such as OCaml and Haskell. ADT have found their ways in languages such as Rust, which allows aggressive optimisations of their memory representation.

This year, we extended our setup to more complex types and a richer compilation algorithm by exhibiting so-called "memory morphisms", which we presented in a Research Report [22] and at the FProper 2024 Workshop. Additionally, Thaïs Baudondefended her thesis defended in November 2024.

8 Bilateral contracts and grants with industry

8.1 Partnertship with the Aniah startup on circuit verification

Participants: Bruno Ferres, Matthieu Moy, Ludovic Henrio, Gabriel Radanne, Oussama Oulkaid.

The CASH team started discussion with the Aniah startup in 2019, to work on verification of electrical properties of circuits at transistor level. We recruited a post-doc (Bruno Ferres) in March 2022, and formalized the collaboration with a bilateral contract (Réf. Inria : 2021-1144), in parallel with a joint internship with LIP, Verimag laboratory and Aniah (Oussama Oulkaid), which led to a CIFRE Ph.D (LIP/Verimag/Aniah) started in October 1^{st} 2022. The collaboration led to the development of a prototype tool, which served a the basis for the re-implementation of the approach in the production tool, and to two articles accepted at the DATE conference plus two ongoing submission.

8.2 CAVOC Project with Inria/Nomadic Labs

Participants: Guilhem Jaber, Gabriel Radanne, Laure Gonnord.

This project aims to develop a *sound and precise static analyzer* for OCaml, that can catch large classes of bugs represented by uncaught exceptions. It will deal with both user-defined exceptions, and built-in ones used to represent *error behaviors*, like the ones triggered by `failwith`, `assert`, or a match failure. Via "assert-failure" detection, it will thus be able to check that invariants annotated by users hold. The analyzer will reason *compositionally* on programs, in order to analyze them at the granularity of a

function or of a module. It will be *sound* in a strong way: if an OCaml module is considered to be correct by the analyzer, then one will have the guarantee that no OCaml code interacting with this module can trigger uncaught exceptions coming from the code of this module. In order to be *precise*, it will take into account the abstraction properties provided by the type system and the module system of the language: local values, abstracted definition of types, parametric polymorphism. The goal being that most of the interactions taken into account correspond to typeable OCaml code.

This project is part of the partnership between Inria and Nomadic Labs, and lead by Guilhem Jaber, from the Inria Team Galinette.

9 Partnerships and cooperations

9.1 International initiatives

9.1.1 Participation in other International Programs

Polytrace Exploratory Action

Participants: Christophe Alias, Keiji Kimura.

Title: Polytrace – Scaling Polyhedral Compilers with Trace Analysis

Partner Institution(s): Waseda University, Japan

Date/Duration: 4 years, until December 2024.

Additionnal info/keywords: Compilers, HPC, Polyhedral Model, Trace Analysis

9.2 International research visitors

9.2.1 Visits of international scientists

Other international visits to the team

Keiji Kimura

Status Professor

Institution of origin: Waseda University

Country: Japan

Dates: 16/12/2024-19/12/2024

Context of the visit: Collaboration, PhD defense of Hugo Thievenaz

Mobility program/type of mobility: research stay

Violet Ka I Pun and Einar Broch Johnsen

Status (professor/assistant professor))

Institution of origin: Univ of applied science(Bergen) and Univ of Oslo

Country: Norway

Dates: 18/11/2024-22/11/2024

Context of the visit: Collaboration

Mobility program/type of mobility: research stay

9.3 National initiatives

PEPR NumPex, ExaSoft Project (WP2, Task 2.3)

Participants: Alec Sadler, Christophe Alias, Thierry Gautier, Xavier Rival, Philippe Clauss.

Title: Polysparse – Compiling Sparse Kernels by Specialization

Partner Institution(s): Inria Paris (X. Rival), Inria Nancy (P. Clauss)

Date/Duration: 6 years, started this year.

Additionnal info/keywords: Compilers, HPC, Polyhedral Model, Sparse Computation

10 Dissemination

10.1 Promoting scientific activities

The team participated in a DECLICS (Dialogues Entre Chercheurs et Lycéens pour les Intéresser à la Construction des Savoirs) meeting with high-school students (Lycée Lycée Jean Puy - Lyon - Roanne) in December 2024.

10.1.1 Scientific events: organisation

General chair, scientific chair

- Ludovic Henrio is member of the steering committee of ICE workshops.
- Cyril Cohen is a member of the steering committee of the ITP conference.

10.1.2 Journal

- Christophe Alias was a reviewer for JPDC, PARCO, SIAM, TETC.
- Gabriel Radanne was an external reviewer for JFP (1 paper)
- Matthieu Moy was a reviewer for TACO and TECS.

10.1.3 Conferences

Member of the conference program committees

- Christophe Aliaswas a PC member for COMPAS'24, HPC Bugs Fest'24
- Yannick Zakowskiwas a PC member for OOPSLA'25, JFLA'25, POPL'24, PPDP'24, JFLA'24

Member of the workshop program committees

- Yannick Zakowskiwas a PC member for CoqWS'24, GALOP'24, CoqPL'24,

Member of the Artifact Evaluation committees

- Nicolas Chappe was a member of the Artifact Evaluation Committee for POPL'24
- Yannick Zakowski was a member of the Artifact Evaluation Committee for ICFP'24

10.1.4 Leadership within the scientific community

- Ludovic Henrio is one of the responsibles, with Frédéric Dabrowski of the GdT CLAP inside the GDR GPL.
- Laure Gonnord is responsible for "Ecole des Jeunes Chercheurs en Programmation" in the GdR GPL.

10.1.5 Scientific expertise

- Christophe Alias is scientific advisor for the XTREMLOGIC startup.

10.1.6 Research administration

- Ludovic Henrio is member of the "commission recherche" of labex Milyon.
- Ludovic Henrio is part of the "équipe de direction" of the LIP labpratory, as a deputy for contracts and international collaborations.
- Ludovic Henrio is "référent de site" for the PIQ program.

10.2 Teaching - Supervision - Juries

10.2.1 Teaching

Licence

- Christophe Alias: "Compilation", INSA CVL 3A, cours+TD, 27h ETD.
- Hugo Thievenaz: "Algorithmique programmation impérative, initiation", L1 UCBL, TP, 18h ETD.
- Hugo Thievenaz: "Bases de l'architecture pour la programmation", L1 UCBL, TP, 24h ETD.
- Alec Sadler: "Systèmes d'exploitation", L2 UCBL, TD/TP, 21h ETD.
- Alec Sadler: "Programmation concurrente et administration système", L3 UCBL, TP, 16.5h ETD.
- Matthieu Moy: "Référent pédagogique", supervising 100 students/year; "Programmation web", L3 UCBL, 19h; "Projet Informatique", L3 UCBL, 9.5h; "Systèmes d'exploitation", L2 UCBL, 25h.
- Nicolas Chappe: "Architecture des ordinateurs", L2 UCBL, 24h TP
- Emanuel Arrighi: "Théorie de la Programmation", L3 ENS, TP
- Emanuel Arrighi: "Algorithmique 1", L3 ENS, TP
- Yannick Zakowski: organisation du "Séminaire du département d'informatique", L3 ENS

Master 1

- Christophe Alias: "Optimisation des applications embarquées", INSA CVL 4A, cours+TD, 27h ETD.
- Christophe Alias: "Compilation", Préparation à l'agrégation d'informatique, ENS-Lyon, cours, 18h ETD.
- Alec Sadler: "Compilation / traduction des programmes", M1 UCBL, TD/TP, 21.5h ETD.
- Thaïs Baudon: TD de compilation, préparation à l'agrégation d'informatique, ENS de Lyon, 10 ETD.
- Matthieu Moy: "Compilation et traduction des programmes", M1 UCBL, responsible, 31h; "Gestion de projet et génie logiciel", M1 UCBL, responsible, 32h; "Projet pour l'Orientation en Master", M1 UCBL, 2 students supervised.

- Hugo Thievenaz: "Compilation / traduction des programmes", M1 UCBL, TD+TP, 22.5h ETD.
- Gabriel Radanne, Yannick Zakowski and Emma Nardino: "Compilation and Analysis" (CAP), ENS-Lyon, Master d'Informatique Fondamentale, cours, 48h CM + 28h TD, 64h ETD.
- Gabriel Radanne: "Projet", M1 ENS-Lyon, 5 students supervised.

10.2.2 Supervision

- Christophe Alias co-advised the PhD thesis of Hugo Thievenaz with Keiji Kimura (Waseda University).
- Christophe Alias advises the PhD thesis of Alec Sadler with the collaboration of Thierry Gautier, Xavier Rival (Inria Paris) and Philippe Clauss (Inria Strasbourg).
- Ludovic Henrio and Yannick Zakowski co-advised the PhD thesis of Nicolas Chappe.
- Ludovic Henrio and Yannick Zakowski co-advise the PhD thesis of Emma Nardino. Gabriel Radanne also collaborates actively scientifically to the PhD thesis.
- Yannick Zakowski co-advises the PhD thesis of Peio Borthelle with Tom Hirschowitz (CNRS, Chambéry) and Guilhem Jaber (Nantes Université).
- Gabriel Radanne and Laure Gonnord co-advised the PhD thesis of Thaïs Baudon.
- Matthieu Moy co-advises the PhD thesis of Oussama Oulkaid with Pascal Raymond (Verimag), Mehdi Khosravian (Aniah), and Bruno Ferres (Verimag).
- Cyril Cohen co-advises the PhD thesis of Quentin Vermande with Yves Bertot (Inria, Sophia-Antipolis)
- Cyril Cohen co-advises the PhD thesis of Vojtěch Štěpančík with Assia Mahboubi (Inria, Nantes).

10.2.3 Defended Ph.D

- Thaïs Baudon defended her PhD thesis entitled "Types algébriques pour le calcul haute-performance" on October 15.
- Nicolas Chappe defended his PhD thesis entitled "A concurrency model based on monadic interpreters: Executable semantics for a concurrent subset of LLVM IR" on November 22.
- Hugo Thievenaz defended his PhD thesis entitled "Scalable trace-based compile-time memory allocation" on December 18.

10.2.4 Juries

- Christophe Alias was a member of the hiring committee for Inria Lyon (CRCN,ISFP).
- Christophe Alias was examiner for the "oral d'informatique du second concours de l'ENS de Lyon" competitive examination.
- Christophe Alias was "correcteur" for the "X/ENS filière PSI" competitive examination.
- Christophe Alias was a specialist member for the CSI of Ugo Battiston, Raphaël Colin, Clément Rosseti (advisor: Philippe Clauss); Tom Hammer (advisors: Vincent Loechner and Stéphane Genaud).
- Ludovic Henrio was reviewer (rapporteur) for the HDR of Frédéric Dabrowski.
- Yannick Zakowski was examiner (examinateur) for the PhD defense of Alexandre Moine.
- Yannick Zakowski was examiner (examinateur) for the PhD defense of Paul Jeanmaire .

10.3 Popularization

10.3.1 Education

- Thaïs Baudon: Conception d'activités débranchées pour la vulgarisation de l'informatique et des mathématiques auprès du grand public, Maison des Mathématiques et de l'Informatique (MMI), 32h ETD.
- Thaïs Baudon: "Maths en Jeans" with Joël Felderhoff et Daniel Hirschkoff
- Matthieu Moy: supervision of "stages de 2nde" (high-school internships).

11 Scientific production

11.1 Major publications

[1] C. Alias and A. Plesco. 'Data-Aware Process Networks'. In: CC 2021 - 30th ACM SIGPLAN International Conference on Compiler Construction. Virtual, South Korea: ACM, 2nd Mar. 2021, pp. 1–11. DOI: 10.1145/3446804.3446847. URL: https://hal.inria.fr/hal-03143777.

[2] T. Baudon, G. Radanne and L. Gonnord. 'Bit-Stealing Made Legal: Compilation for Custom Memory Representations Of Algebraic Data Types'. In: *Proceedings of the ACM on Programming Languages*. ICFP 2023. ICFP. Seattle (USA), United States, 4th Sept. 2023. DOI: 10.1145/3607858. URL: https://inria.hal.science/hal-04165615.

[3] N. Chappe, P. He, L. Henrio, Y. Zakowski and S. Zdancewic. 'Choice Trees: Representing Non-deterministic, Recursive, and Impure Programs in Coq'. In: *Proceedings of the ACM on Programming Languages* (15th Jan. 2023), pp. 1–31. DOI: 10.1145/3571254. URL: https://hal.science/hal-03886910.

[4] N. Chappe, L. Henrio, A. Maillé, M. Moy and H. Renaud. 'An Optimised Flow for Futures: From Theory to Practice'. In: *The Art, Science, and Engineering of Programming* 6.1 (15th July 2021), pp. 1–41. DOI: 10.22152/programming-journal.org/2022/6/3. URL: https://hal.inria.fr/hal-03440766.

[5] C. C. Din, R. Hähnle, L. Henrio, E. B. Johnsen, V. K. I. Pun and S. L. T. Tarifa. 'Locally Abstract, Globally Concrete Semantics of Concurrent Programming Languages'. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 46.1 (29th Mar. 2024), pp. 1–58. DOI: 10.1145/3648439. URL: https://hal.science/hal-04732946.

[6] L. Gonnord, L. Henrio, L. Morel and G. Radanne. 'A Survey on Parallelism and Determinism'. In: *ACM Computing Surveys* (27th Sept. 2022). DOI: 10.1145/3564529. URL: https://hal.inria.fr/hal-03828497.

[7] R. Hähnle and L. Henrio. 'Provably Fair Cooperative Scheduling'. In: *The Art, Science, and Engineering of Programming* 8.2 (15th Oct. 2023). DOI: 10.22152/programming-journal.org/2024/8/6. URL: https://hal.science/hal-04372450.

[8] O. Oulkaid, B. Ferres, M. Moy, P. Raymond, M. Khosravian, L. Henrio and G. Radanne. 'A Transistor Level Relational Semantics for Electrical Rule Checking by SMT Solving'. In: *https://ieeexplore.ieee.org/document/10546537*. Design, Automation and Test in Europe Conference. Valencia, Spain: IEEE, 2024, pp. 1–6. DOI: 10.23919/DATE58400.2024.10546537. URL: https://hal.science/hal-04527225.

[9] Y. Zakowski, C. Beck, I. Yoon, I. Zaichuk, V. Zaliva and S. Zdancewic. 'Modular, compositional, and executable formal semantics for LLVM IR'. In: *Proceedings of the ACM on Programming Languages* 5.ICFP (22nd Aug. 2021), pp. 1–30. DOI: 10.1145/3473572. URL: https://hal.archives-ouvertes.fr/hal-03525711.

11.2 Publications of the year

International journals

[10] C. Beck, I. Yoon, H. Chen, Y. Zakowski and S. Zdancewic. 'A Two-Phase Infinite/Finite Low-Level Memory Model: Reconciling Integer–Pointer Casts, Finite Space, and undef at the LLVM IR Level of Abstraction'. In: *Proceedings of the ACM on Programming Languages* 8.ICFP (15th Aug. 2024), pp. 789–817. DOI: 10.1145/3674652. URL: https://hal.science/hal-04691859 (cit. on p. 15).

[11] C. Blaudeau, D. Rémy and G. Radanne. 'Avoiding Signature Avoidance in ML Modules with Zippers'. In: *Proceedings of the ACM on Programming Languages* POPL.9 (22nd Jan. 2025). DOI: 10.1145/3704902. URL: https://inria.hal.science/hal-04801582 (cit. on p. 16).

[12] C. Blaudeau, D. Rémy and G. Radanne. 'Fulfilling OCaml Modules with Transparency'. In: *Proceedings of the ACM on Programming Languages* 8.OOPSLA1 (29th Apr. 2024), pp. 194–222. DOI: 10.1145/3649818. URL: https://inria.hal.science/hal-04794404 (cit. on p. 16).

[13] H. Coullon, L. Henrio, F. Loulergue and S. Robillard. 'Component-Based Distributed Software Reconfiguration: a Verification-Oriented Survey'. In: *ACM Computing Surveys* 56.1 (Jan. 2024), pp. 1–37. DOI: 10.1145/3595376. URL: https://inria.hal.science/hal-04067909.

[14] C. C. Din, R. Hähnle, L. Henrio, E. B. Johnsen, V. K. I. Pun and S. L. T. Tarifa. 'Locally Abstract, Globally Concrete Semantics of Concurrent Programming Languages'. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 46.1 (29th Mar. 2024), pp. 1–58. DOI: 10.1145/3648439. URL: https://hal.science/hal-04732946.

[15] S. Michelland, Y. Zakowski and L. Gonnord. 'Abstract Interpreters: A Monadic Approach to Modular Verification'. In: *Proceedings of the ACM on Programming Languages* 8.ICFP (15th Aug. 2024). DOI: 10.1145/3674646. URL: https://hal.science/hal-04628727 (cit. on p. 17).

International peer-reviewed conferences

[16] C. Cohen, E. Crance and A. Mahboubi. 'Trocq: Proof Transfer for Free, With or Without Univalence'. In: *Lecture Notes in Computer Science*. ESOP 2024 - 33rd European Symposium on Programming. 14576. Luxembourg, Luxembourg: Springer, 2024, pp. 239–268. URL: https://hal.science/hal-04177913 (cit. on p. 14).

[17] O. Oulkaid, B. Ferres, M. Moy, P. Raymond, M. Khosravian, L. Henrio and G. Radanne. 'A Transistor Level Relational Semantics for Electrical Rule Checking by SMT Solving'. In: *https://ieeexplore.ieee.org/document/10546537*. Design, Automation and Test in Europe Conference. Valencia, Spain: IEEE, 2024, pp. 1–6. DOI: 10.23919/DATE58400.2024.10546537. URL: https://hal.science/hal-04527225 (cit. on p. 16).

Conferences without proceedings

[18] E. Arrighi and G. Radanne. 'Light-speed type unification modulo isomorphisms'. In: ML Workshop 2024. Milan, Italy, 6th Sept. 2024. URL: https://inria.hal.science/hal-04794390 (cit. on p. 17).

[19] W. Du, L. Henrio and G. Radanne. 'Distributed Actors in OCaml'. In: OCaml Workshop. Vol. OCaml. 2024. Milan, Italy, 7th Sept. 2024. URL: https://hal.science/hal-04794441 (cit. on p. 13).

[20] V. Gachet, L. Henrio and G. Radanne. 'Tail Modulo Async/Await - Extended Abstract'. In: FPROPER. Vol. FPROPER. 2024. Milan, Italy, 6th Sept. 2024. URL: https://hal.science/hal-04794434 (cit. on p. 13).

Scientific book chapters

[21] M. Andrieux, L. Henrio and G. Radanne. 'Active Objects based on Algebraic Effects'. In: *Active Object Languages: Current Research Trends*. Vol. 14360. Lecture Notes in Computer Science. 2024, pp. 3–36. DOI: 10.1007/978-3-031-51060-1_1. URL: https://hal.science/hal-04388798.

Reports & preprints

[22] T. Baudon, G. Radanne and L. Gonnord. *Compiling Morphisms of Algebraic Data Types*. June 2024. URL: https://hal.science/hal-04601882 (cit. on p. 18).

[23] P. Borthelle, T. Hirschowitz, G. Jaber and Y. Zakowski. *An abstract, certified account of operational game semantics*. 22nd May 2024. URL: https://hal.science/hal-04583895 (cit. on p. 14).

[24] N. Chappe, L. Henrio and Y. Zakowski. *A concurrency model based on monadic interpreters: executable semantics for a concurrent subset of LLVM IR*. 30th May 2024. URL: https://hal.science/hal-04594073 (cit. on p. 16).

[25] S. Michelland, Y. Zakowski and L. Gonnord. *Abstract Interpreters: a Monadic Approach to Modular Verification (DRAFT)*. 10th Jan. 2024. URL: https://inria.hal.science/hal-04385725.

Software

[26] [SW] G. Radanne, *Dowsing* 18th June 2024. LIC: MIT License. HAL: ⟨hal-04616085⟩, URL: https://hal.science/hal-04616085, VCS: https://github.com/Drup/dowsing/, SWHID: ⟨swh:1:dir:b93e4b399fd773bfb9397f170a8649f4552b191b⟩ (cit. on p. 17).

11.3 Cited publications

[27] L. Xia, Y. Zakowski, P. He, C. Hur, G. Malecha, B. C. Pierce and S. Zdancewic. 'Interaction Trees'. In: *Proceedings of the ACM on Programming Languages* 4.POPL (2020). DOI: 10.1145/3371119 (cit. on p. 15).

[28] Y. Zakowski, C. Beck, I. Yoon, I. Zaichuk, V. Zaliva and S. Zdancewic. 'Modular, compositional, and executable formal semantics for LLVM IR'. In: *Proceedings of the ACM on Programming Languages* 5.ICFP (Aug. 2021), pp. 1–30. DOI: 10.1145/3473572. URL: https://hal.science/hal-03525711 (cit. on p. 15).