IN PARTNERSHIP WITH:
IMT Atlantique Bretagne-Pays de la Loire

Université Nantes

Activity Report 2018

Project-Team GALLINETTE

Gallinette: developing a new generation of proof assistants

IN COLLABORATION WITH: Laboratoire des Sciences du numerique de Nantes

RESEARCH CENTER
Rennes - Bretagne-Atlantique

THEME
Proofs and Verification

Table of contents

Project-Team GALLINETTE

Creation of the Team: 2017 May 01, updated into Project-Team: 2018 June 01

Keywords:

Computer Science and Digital Science:

A2.1.1. - Semantics of programming languages
A2.1.2. - Imperative programming
A2.1.3. - Object-oriented programming
A2.1.4. - Functional programming
A2.1.11. - Proof languages
A2.2.3. - Memory management
A2.4.3. - Proofs
A7.2.3. - Interactive Theorem Proving
A7.2.4. - Mechanized Formalization of Mathematics
A8.4. - Computer Algebra

Other Research Topics and Application Domains:

B6.1. - Software industry

1. Team, Visitors, External Collaborators

Research Scientists

Assia Mahboubi [Inria, Researcher]
Guillaume Munch-Maccagnoni [Inria, Researcher]
Pierre-Marie Pédrot [Inria, Researcher, from Oct 2018]
Nicolas Tabareau [Team leader, Inria, Researcher, HDR]

Faculty Members

Julien Cohen [Univ de Nantes, Associate Professor]
Rémi Douence [IMT Atlantique, Associate Professor, HDR]
Hervé Grall [IMT Atlantique, Associate Professor]
Guilhem Jaber [Univ de Nantes, Associate Professor from Sep 2018]

Post-Doctoral Fellows

Danil Annenkov [Inria, until Nov 2018]
Eric Finster [Inria]
Marie Kerjean [Inria, from Nov 2018]
Maxime Lucas [Inria, from Sep 2018]
Étienne Miquey [Inria]
Pierre Vial [Inria]

PhD Students

Antoine Allioux [Inria]
Simon Boulier [Ecole normale supérieure de Rennes, until Aug 2018]
Gaëtan Gilbert [IMT Atlantique]
Ambroise Lafont [IMT Atlantique]
Xavier Montillet [Univ de Nantes]
Théo Winterhalter [Univ de Nantes]
Igor Zhirkov [Armines]

Technical staff
Simon Boulier [Inria, from Oct 2018]

Interns
Antoine Defourné [Inria, from Apr 2018 until Aug 2018]
Loic Pujet [Ecole Normale Supérieure Paris, from Mar 2018 until Jul 2018]

Administrative Assistant
Anne-Claire Binétruy [Inria]

Visiting Scientist
Ambrus Kaposi [Eötvös Loránd University, Budapest, Hungary, from Apr 2018 until Jul 2018]

2. Overall Objectives

2.1. Overall Objectives

The EPI Gallinette aims at developing a new generation of proof assistants, with the belief that practical experiments must go in pair with foundational investigations:

- The goal is to advance proof assistants both as certified programming languages and mechanised logical systems. Advanced programming and mathematical paradigms must be integrated, notably dependent types and effects. The distinctive approach is to implement new programming and logical paradigms on top of Coq by considering the latter as a target language for compilation.
- The aim of foundational investigations is to extend the boundaries of the Curry-Howard correspondence. It is seen both as providing foundations for programming languages and logic, and as a purveyor of techniques essential to the development of proof assistants. Under this perspective, the development of proof assistants is seen as a total experiment using the correspondence in every aspect: programming languages, type theory, proof theory, rewriting and algebra.

3. Research Program

3.1. Scientific Context

Software quality is a requirement that is becoming more and more prevalent, by now far exceeding the traditional scope of embedded systems. The development of tools to construct software that respects a given specification is a major challenge facing computer science. *Proof assistants* such as Coq [50] provide a formal method whose central innovation is to produce *certified programs* by transforming the very activity of programming. Programming and proving are merged into a single development activity, informed by an elegant but rigid mathematical theory inspired by the correspondence between programming, logic and algebra: the *Curry-Howard correspondence*. For the certification of programs, this approach has shown its efficiency in the development of important pieces of certified software such as the C compiler of the CompCert project [79]. The extracted CompCert compiler is reliable and efficient, running only 15% slower than GCC 4 at optimisation level 2 (`gcc -O2`), a level of optimisation that was considered before to be highly unreliable.

Proof assistants can also be used to *formalise mathematical theories*: they not only provide a means of representing mathematical theories in a form amenable to computer processing, but their internal logic provides a language for reasoning about such theories. In the last decade, proof assistants have been used to verify extremely large and complicated proofs of recent mathematical results, sometimes requiring either intensive computations [61], [65] or intricate combinations of a multitude of mathematical theories [60]. But formalised mathematics is more than just proof checking and proof assistants can help with the organisation mathematical knowledge or even with the discovery of new constructions and proofs.

Unfortunately, the rigidity of the theory behind proof assistants impedes their expressiveness both as programming languages and as logical systems. For instance, a program extracted from Coq only uses a purely functional subset of OCaml, leaving behind important means of expression such as side-effects and objects. Limitations also appears in the formalisation of advanced mathematics: proof assistants do not cope well with classical axioms such as excluded middle and choice which are sometimes used crucially. The fact of the matter is that the development of proof assistants cannot be dissociated from a reflection on the nature of programs and proofs coming from the Curry-Howard correspondence. In the EPC Gallinette, we propose to address several drawbacks of proof assistants by pushing the boundaries of this correspondence.

In the 1970's, the Curry-Howard correspondence was seen as a perfect match between functional programs, intuitionistic logic, and Cartesian closed categories. It received several generalisations over the decades, and now it is more widely understood as a fertile correspondence between computation, logic, and algebra. Nowadays, the view of the Curry-Howard correspondence has evolved from a perfect match to a collection of theories meant to explain similar structures at work in logic and computation, underpinned by mathematical abstractions. By relaxing the requirement of a perfect match between programs and proofs, and instead emphasising the common foundations of both, the insights of the Curry-Howard correspondence may be extended to domains for which the requirements of programming and mathematics may in fact be quite different.

Consider the following two major theories of the past decades, which were until recently thought to be irreconcilable:

- **(Martin-Löf) Type theory:** introduced by Martin-Löf in 1971, this formalism [86] is both a programming language and a logical system. The central ingredient is the use of *dependent types* to allow fine-grained invariants to be expressed in program types. In 1985, Coquand and Huet developed a similar system called the *calculus of constructions*, which served as logical foundation of the first implementation of Coq. This kind of systems is still under active development, especially with the recent advent of homotopy type theory (HoTT) [108] which gives a new point of view on types and the notion of equality in type theory.
- **The theory of effects:** starting in the 1980's, Moggi [91] and Girard [58] put forward monads and co-monads as describing various compositional notions of computation. In this theory, programs can have side-effects (state, exceptions, input-output), logics can be non-intuitionistic (linear, classical), and different computational universes can interact (modal logics). Recently, the safe and automatic management of resources has also seen a coming of age (Rust, Modern C++) confirming the importance of linear logic for various programming concepts. It is now understood that the characteristic feature of the theory of effects is sensitivity to *evaluation order*, in contrast with type theory which is built around the assumption that evaluation order is irrelevant.

We now outline a series of scientific challenges aimed at understanding of type theory, effects, and their combination.

More precisely, three key axes of improvement have been identified:

1. Making the notion of equality closer to what is usually assumed when doing proofs on black board, with a balance between irrelevant equality for simple structures and equality up-to equivalences for more complex ones (Section 3.2). Such a notion of equality should allow one to implement traditional model transformations that enhance the logical power of the proof assistant using distinct compilation phases.
2. Advancing the foundations of effects within the Curry-Howard approach. The objective is to pave the way for the integration of effects in proof assistants and to prototype the corresponding implementation. This integration should allow for not only certified programming with effects, but also the expression of more powerful logics (Section 3.3).
3. Making more programming features (notably, object polymorphism) available in proof assistants, in order to scale to practical-sized developments. The objective is to enable programming styles closer to common practices. One of the key challenges here is to leverage gradual typing to dependent programming (Section 3.4).

To validate the new paradigms, we propose in Section 3.5 three particular application fields in which members of the team already have a strong expertise: code refactoring, constraint programming and symbolic computation.

3.2. Enhance the computational and logical power of proof assistants

The democratisation of proof assistants based on type theory has likely been impeded one central problem: the mismatch between the conception of equality in mathematics and its formalisation in type theory. Indeed, some basic principles that are used implicitly in mathematics—such as Church's principle of propositional extensionality, which says that two propositions are equal when they are logically equivalent—are not derivable in type theory. Even more problematically, from a computer science point of view, the basic concept of two functions being equal when they are equal at every "point" of their domain is also not derivable: rather, it must be added as an additional axiom. Of course, these principles are consistent with type theory so that working under the corresponding additional assumptions is safe. But the use of these assumptions in a definition potentially clutters its computational behaviour: since axioms are computational black boxes, computation gets stuck at the points of the code where they have been used.

We propose to investigate how expressive logical transformations such as forcing [71] and sheaf construction might be used to enhance the computational and logical power of proof assistants—with a particular emphasis on their implementation in the Coq proof assistant by the means of effective translations (or compilation phases). One of the main topics of this task, in connection to the ERC project CoqHoTT, is the integration in Coq of new concepts inspired by homotopy type theory [108] such as the univalence principle, and higher inductive types.

3.2.1. A definitional proof-irrelevant version of Coq.

In the Coq proof assistant, the sort $\mathbf{Prop}$ stands for the universe of types which are propositions. That is, when a term P has type $\mathbf{Prop}$, the only relevant fact is whether P is inhabited (that is true) or not (that is false). This property, known as *proof irrelevance,* can be expressed formally as: $\forall x\ y : P, x = y$. Originally, the *raison d'être* of the sort $\mathbf{Prop}$ was to characterise types with no computational meaning with the intention that terms of such types could be erased upon extraction. However, the assumption that every element of $\mathbf{Prop}$ should be proof irrelevant has never been integrated to the system. Indeed, in Coq, proof irrelevance for the sort $\mathbf{Prop}$ is not incorporated into the theory: it is only compatible with it, in the sense that its assumption does not give rise to an inconsistent theory. In fact, the exact status of the sort $\mathbf{Prop}$ in Coq has never been entirely clarified, which explains in part this lack of integration. Homotopy type theory brings fresh thinking on this issue and suggests turning $\mathbf{Prop}$ into the collection of terms that a certain static inference procedure tags as proof irrelevant. The goal of this task is to integrate this insight in the Coq system and to implement a definitional proof-irrelevant version of the sort $\mathbf{Prop}$.

3.2.2. Extend the Coq proof assistant with a computational version of univalence

The univalence principle is becoming widely accepted as a very promising avenue to provide new foundations for mathematics and type theory. However, this principle has not yet been incorporated into a proof assistant. Indeed, the very mathematical structures (known as ∞-groupoids) motivating the theory remain to this day an active area of research. Moreover, a correct and decidable type checking procedure for the whole theory raises both computational complexity and logical coherence issues. Observational type theory [33], as implemented in Epigram, provides a first-stage approximation to homotopy type theory, but only deals with functional extensionality and does not capture univalence. Coquand and his collaborators have obtained significant results on the computational meaning of univalence using cubical sets [40], [46]. Bickford has initiated a promising formalisation work [1] in the NuPRL system. However, a complete formalisation in intensional type theory remains an open problem.

[1] http://www.nuprl.org/wip/Mathematics/cubical!type!theory/index.html

Hence a major objective is to achieve a complete internalisation of univalence in intensional type theory, including an integration to a new version of Coq. We will strive to keep compatibility with previous versions, in particular from a performance point of view. Indeed, the additional complexity of homotopy type theory should not induce an overhead in the type checking procedure used by the software if we want our new framework to become rapidly adopted by the community. Concretely, we will make sure that the compilation time of Coq's Standard Library will be of the same order of magnitude.

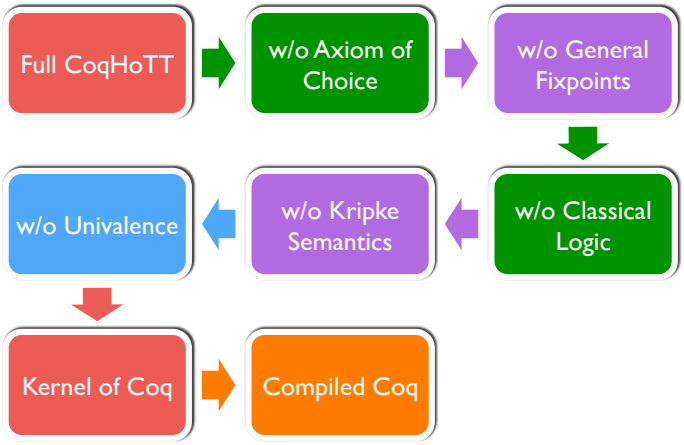

Figure 1. Multiple compilation phases to increase the logical and computational power of Coq.

3.2.3. Extend the logical power of type theory without axioms in a modular way

Extending the power of a logic using model transformations (*e.g.,* forcing transformation [72], [71] or the sheaf construction [101]) is a classic topic of mathematical logic [47], [77]. However, these ideas have not been much investigated in the setting of type theory, even though they may provide a useful framework for extending the logical power of proof assistant in a modular way. There is a good reason for this: with a syntactic notion of equality, the underlying structure of type theory does not conform to the structure of topos used in mathematical logic. A direct incorporation of the standard techniques is therefore not possible. However, a univalent notion of equality brings type theory closer to the required algebraic structure, as it corresponds to the notion of ∞-topos recently studied by Lurie [84]. The goal of this task is to revisit model transformations in the light of the univalence principle, and to obtain in this way new internal transformations in type theory which can in turn be seen as compilation phases. The general notion of an internal syntactical translation has already been investigated in the team [41].

3.2.4. Methodology: Extending type theory with different compilation phases

The Gallinette project advocates the use of distinct compilation phases as a methodology for the design of a new generation of proof assistants featuring modular extensions of a core logic. The essence of a compiler is the separation of the complexity of a translation process into modular stages, and the organization of their re-composition. This idea finds a natural application in the design of complex proof assistants (Figure 1). For instance, the definition of type classes in Coq follows this pattern, and is morally given by the means of a translation into a type-class free kernel. More recently, a similar approach by compilation stages, using the forcing transformation, was used to relax the strict positivity condition guarding inductive types [72], [71]. We believe that this flavour of compilation-based strategies offers a promising direction of investigation for the propose of defining a decidable type checking algorithm for HoTT.

3.3. Semantic and logical foundations for effects in proof assistants based on type theory

We propose the incorporation of effects in the theory of proof assistants at a foundational level. Not only would this allow for certified programming with effects, but it would moreover have implications for both semantics and logic.

We mean *effects* in a broad sense that encompasses both Moggi's monads [91] and Girard's linear logic [58]. These two seminal works have given rise to respective theories of effects (monads) and resources (co-monads). Recent advances, however have unified these two lines of thought: it is now clear that the defining feature of effects, in the broad sense, is sensitivity to evaluation order [80], [51].

In contrast, the type theory that forms the foundations of proof assistants is based on pure λ calculus and is built on the assumption that evaluation order is irrelevant. Evaluation order is therefore the blind spot of type theory. In Moggi [92], integrating the dependent types of type theory with monads is *"the next difficult step [...] currently under investigation"*.

Any realistic program contains effects: state, exceptions, input-output. More generally, evaluation order may simply be important for complexity reasons. With this in mind, many works have focused on certified programming with effects: notably Ynot [96], and more recently $F^{\star}$ [106] and Idris [42], which propose various ways for encapsulating effects and restricting the dependency of types on effectful terms. Effects are either specialised, such as the monads with Hoare-style pre- and post-conditions found in Ynot or $F^{\star}$, or more general, such as the algebraic effects implemented in Idris. But whereas there are several experiments and projects pursuing the certification of programs with effects, each making its own choices on how effects and dependency should be merged, there is on the other hand a deficit of logical and semantic investigations.

We propose to develop the foundations of a type theory with effects taking into account the logical and semantic aspects, and to study their practical and theoretical consequences. A type theory that integrates effects would have logical, algebraic and computational implications when viewed through the Curry-Howard correspondence. For instance, effects such as control operators establish a link with classical proof theory [63]. Indeed, control operators provide computational interpretations of type isomorphisms such as $A \cong \neg\neg A$ and $\neg\forall x A \cong \exists x \neg A$ (e.g. [93]), whereas the conventional wisdom of type theory holds that such axioms are non-constructive (this is for instance the point of view that has been advocated so far in homotopy type theory [108]). Another example of an effect with logical content is state (more precisely memoization) which is used to provide constructive content to the classical dependent axiom of choice [39], [75], [67]. In the long term, a whole body of literature on the constructive content of classical proofs is to be explored and integrated, providing rich sources of inspiration: Kohlenbach's proof mining [74] and Simpson's reverse mathematics [104], for instance, are certainly interesting to investigate from the Curry-Howard perspective.

The goal is to develop a type theory with effects that accounts both for practical experiments in certified programming, and for clues from denotational semantics and logical phenomena, in a unified setting.

3.3.1. Models for integrating effects with dependent types

A crucial step is the integration of dependent types with effects, a topic which has remained *"currently under investigation"* [92] ever since the beginning. The difficulty resides in expressing the dependency of types on terms that can perform side-effects during the computation. On the side of denotational semantics, several extensions of categorical models for effects with dependent types have been proposed [30], [109] using axioms that should correspond to restrictions in terms of expressivity but whose practical implications, however, are not immediately transparent. On the side of logical approaches [67], [68], [78], [90], one first considers a drastic restriction to terms that do not compute, which is then relaxed by semantic means. On the side of systems for certified programming such as $F^{\star}$, the type system ensures that types only depend on pure and terminating terms.

Thus, the recurring idea is to introduce restrictions on the dependency in order to establish an encapsulation of effects. In our approach, we seek a principled description of this idea by developing the concept of *semantic*

value (thunkables, linears) which arose from foundational considerations [57], [103], [94] and whose relevance was highlighted in recent works [81], [100]. The novel aspect of our approach is to seek a proper extension of type theory which would provide foundations for a classical type theory with axiom of choice in the style of Herbelin [67], but which moreover could be generalised to effects other than just control by exploiting an abstract and adaptable notion of semantic value.

3.3.2. Intuitionistic depolarisation

In our view, the common idea that evaluation order does not matter for pure and termination computations should serve as a bridge between our proposals for dependent types in the presence of effects and traditional type theory. Building on the previous goal, we aim to study the relationship between semantic values, purity, and parametricity theorems [102], [59]. Our goal is to characterise parametricity as a form of intuitionistic *depolarisation* following the method by which the first game model of full linear logic was given (Melliès [87], [88]). We have two expected outcomes in mind: enriching type theory with intensional content without losing its properties, and giving an explanation of the dependent types in the style of Idris and $F^{\star}$ where purity- and termination-checking play a role.

3.3.3. Developing the rewriting theory of calculi with effects

An integrated type theory with effects requires an understanding of evaluation order from the point of view of rewriting. For instance, rewriting properties can entail the decidability of some conversions, allowing the automation of equational reasoning in types [28]. They can also provide proofs of computational consistency (that terms are not all equivalent) by showing that extending calculi with new constructs is conservative [105]. In our approach, the λ-calculus is replaced by a calculus modelling the evaluation in an abstract machine [52]. We have shown how this approach generalises the previous semantic and proof-theoretic approaches [34], [80], [82], and overcomes their shortcomings [95].

One goal is to prove computational consistency or decidability of conversions purely using advanced rewriting techniques following a technique introduced in [105]. Another goal is the characterisation of weak reductions: extensions of the operational semantics to terms with free variables that preserve termination, whose iteration is equivalent to strong reduction [29], [55]. We aim to show that such properties derive from generic theorems of higher-order rewriting [111], so that weak reduction can easily be generalised to richer systems with effects.

3.3.4. Direct models and categorical coherence

Proof theory and rewriting are a source of *coherence theorems* in category theory, which show how calculations in a category can be simplified with an embedding into a structure with stronger properties [85], [76]. We aim to explore such results for categorical models of effects [80], [51]. Our key insight is to consider the reflection between *indirect and direct models* [57], [94] as a coherence theorem: it allows us to embed the traditional models of effects into structures for which the rewriting and proof-theoretic techniques from the previous section are effective.

Building on this, we are further interested in connecting operational semantics to 2-category theory, in which a second dimension is traditionally considered for modelling conversions of programs rather than equivalences. This idea has been successfully applied for the λ-calculus [73], [69] but does not scale yet to more realistic models of computation. In our approach, it has already been noticed that the expected symmetries coming from categorical dualities are better represented, motivating a new investigation into this long-standing question.

3.3.5. Models of effects and resources

The unified theory of effects and resources [51] prompts an investigation into the semantics of safe and automatic resource management, in the style of Modern C++ and Rust. Our goal is to show how advanced semantics of effects, resources, and their combination arise by assembling elementary blocks, pursuing the methodology applied by Melliès and Tabareau in the context of continuations [89]. For instance, by combining control flow (exceptions, return) with linearity allows us to describe in a precise way the "Resource Acquisition Is Initialisation" idiom in which the resource safety is ensured with scope-based destructors. A further step would be to reconstruct uniqueness types and borrowing using similar ideas.

3.4. Language extensions for the scaling of proof assistants

The development of tools to construct software systems that respect a given specification is a major challenge of current and future research in computer science. Certified programming with dependent types has recently attracted a lot of interest, and Coq is the *de facto* standard for such endeavours, with an increasing number of users, pedagogical resources, and large-scale projects. Nevertheless, significant work remains to be done to make Coq more usable from a software engineering point of view. The Gallinette team proposes to make progress on three lines of work: (i) the development of gradual certified programming, (ii) the integration of imperative features and object polymorphism in Coq, and (iii) the development of robust tactics for proof engineering for the scaling of formalised libraries.

3.4.1. Gradual Certified Programming

One of the main issues faced by a programmer starting to internalise in a proof assistant code written in a more permissive world is that type theory is constrained by a strict type discipline which lacks flexibility. Concretely, as soon that you start giving more a precise type/specification to a function, the rest of the code interacting with this functions needs to be more precise too. To address this issue, the Gallinette team will put strong efforts into the development of gradual typing in type theory to allow progressive integration of code that comes from a more permissive world.

Indeed, on the way to full verification, programmers can take advantage of a gradual approach in which some properties are simply asserted instead of proven, subject to dynamic verification. Tabareau and Tanter have made preliminary progress in this direction [107]. This work, however, suffers from a number of limitations, the most important being the lack of a mechanism for handling the possibility of runtime errors within Coq. Instead of relying on axioms, this project will explore the application of Section 3.3 to embed effects in Coq. This way, instead of postulating axioms for parts of the development that are too hard/marginal to be dealt with, the system adds dynamic checks. Then, after extraction, we get a program that corresponds to the initial program but with dynamic check for parts that have not been proven, ensuring that the program will raise an error instead of going outside its specification.

This will yield new foundations of gradual certified programming, both more expressive and practical. We will also study how to integrate previous techniques with the extraction mechanism of Coq programs to OCaml, in order to exploit the exception mechanism of OCaml.

3.4.2. Imperative features and object polymorphism in the Coq proof assistant

3.4.2.1. Imperative features.

Abstract data types (ADTs) become useful as the size of programs grows since they provide for a modular approach, allowing abstractions about data to be expressed and then instantiated. Moreover, ADTs are natural concepts in the calculus of inductive constructions. But while it is easy to declare an ADT, it is often difficult to implement an efficient one. Compare this situation with, for example, Okasaki's purely functional data structures [97] which implement ADTs like queues in languages with imperative features. Of course, Okasaki's queues enforce some additional properties for free, such as persistence, but the programmer may prefer to use and to study a simpler implementation without those additional properties. Also in certified symbolic computation (see 3.5.3), an efficient functional implementation of ADTs is often not available, and efficiency is a major challenge in this area. Relying on the theoretical work done in 3.3, we will equip Coq with imperative features and we will demonstrate how they can be used to provide efficient implementations of ADTs. However, it is also often the case that imperative implementation are hard-to-reason-on, requiring for instance the use of separation logic. But in that case, we could take benefice of recent works on integration of separation logic in the Coq proof assistant and in particular the Iris project http://iris-project.org/.

3.4.2.2. Object polymorphism.

Object-oriented programming has evolved since its foundation based on the representation of computations as an exchange of messages between objects. In modern programming languages like Scala, which aims at a synthesis between object-oriented and functional programming, object-orientation concretely results in the use of hierarchies of interfaces ordered by the subtyping relation and the definition of interface implementations

that can interoperate. As observed by Cook and Aldrich [49], [32], interoperability can be considered as the essential feature of objects and is a requirement for many modern frameworks and ecosystems: it means that two different implementations of the same interface can interoperate.

Our objective is to provide a representation of object-oriented programs, by focusing on subtyping and interoperability.

For subtyping, the natural solution in type theory is coercive subtyping [83], as implemented in Coq, with an explicit operator for coercions. This should lead to a shallow embedding, but has limitations: indeed, while it allows subtyping to be faithfully represented, it does not provide a direct means to represent union and intersection types, which are often associated with subtyping (for instance intersection types are present in Scala). A more ambitious solution would be to resort to subsumptive subtyping (or semantic subtyping [56]): in its more general form, a type algebra is extended with boolean operations (union, intersection, complementing) to get a boolean algebra with operators (the original type constructors). Subtying is then interpreted as the natural partial order of the boolean algebra.

We propose to use the type class machinery of Coq to implement semantic subtyping for dependent type theory. Using type class resolution, we can emulate inference rules of subsumptive subtyping without modifying Coq internally. This has also another advantage. As subsumptive subtyping for dependent types should be undecidable in general, using type class resolution allows for an incomplete yet extensible decision procedure.

3.4.3. Robust tactics for proof engineering for the scaling of formalised libraries

When developing certified software, a major part of the effort is spent not only on writing proof scripts, but on *rewriting* them, either for the purpose of code maintenance or because of more significant changes in the base definitions. Regrettably, proof scripts suffer more often than not from a bad programming style, and too many proof developers casually neglect the most elementary principles of well-behaved programmers. As a result, many proof scripts are very brittle, user-defined tactics are often difficult to extend, and sometimes even lack a clear specification. Formal libraries are thus generally very fragile pieces of software. One reason for this unfortunate situation is that proof engineering is very badly served by the tools currently available to the users of the Coq proof assistant, starting with its tactic language. One objective of the Gallinette team is to develop better tools to write proof scripts.

Completing and maintaining a large corpus of formalised mathematics requires a well-designed tactic language. This language should both accommodate the possible specific needs of the theories at stake, and help with diagnostics at refactoring time. Coq's tactic language is in fact two-leveled. First, it includes a basic tactic language, to organise the deductive steps in a proof script and to perform the elementary bureaucracy. Its second layer is a meta-programming language, which allows user to defined their own new tactics at toplevel. Our first direction of work consists in the investigation of the appropriate features of the *basic tactic language*. For instance, the design of the Ssreflect tactic language, and its support for the small scale reflection methodology [62], has been a key ingredient in at least two large scale formalisation endeavours: the Four Colour Theorem [61] and of the Odd Order Theorem [60]. Building on our experience with the Ssreflect tactic language, we will contribute to the ongoing work on the basic tactic language for Coq. The second objective of this task is to contribute to the design of a *typed tactic language*. In particular, we will build on the work of Ziliani and his collaborators [110], extending it with reasoning about the effects that tactics have on the "state of a proof" (e.g. number of sub-goals, metavariables in context). We will also develop a novel approach for incremental type checking of proof scripts, so that programmers gain access to a richer discovery- engineering interaction with the proof assistant.

3.5. Practical experiments

The first three axes of the EPC Gallinette aim at developing a new generation of proof assistants. But we strongly believe that foundational investigations must go hand in hand with practical experiments. Therefore, we expect to benefit from existing expertise and collaborations in the team to experiment our extensions of Coq on real world developments. It should be noticed that those practical experiments are strongly guided by the deep history of research on software engineering of team members.

3.5.1. Certified Code Refactoring

In the context of refactoring of C programs, we intend to formalise program transformations that are written in an imperative style to test the usability of our addition of effects in the proof assistant. This subject has been chosen based on the competence of members of the team.

We are currently working on the formalisation of refactoring tools in Coq [45]. Automatic refactoring of programs in industrial languages is difficult because of the large number of potential interactions between language features that are difficult to predict and to test. Indeed, all available refactoring tools suffer from bugs : they fail to ensure that the generated program has the same behaviour as the input program. To cope with that difficulty, we have chosen to build a refactoring tool with Coq : a program transformation is written in the Coq programming language, then proven correct on all possible inputs, and then an OCaml executable program is generated by the platform. We rely on the CompCert C formalisation of the C language. CompCert is currently the most complete formalisation of an industrial language, which justifies that choice. We have three goals in that project :

- Build a refactoring tool that programmers can rely on and make it available in a popular platform (such as Eclipse, IntelliJ or Frama-C).
- Explore large, drastic program transformations such as replacing a design architecture for an other one, by applying a sequence of small refactoring operations (as we have done for Java and Haskell programs before [48], [44], [31]), while ensuring behaviour preservation.
- Explore the use of enhancements of proof systems on large developments. For instance, refactoring tools are usually developed in the imperative/object paradigm, so the extension of Coq with side effects or with object features proposed in the team can find a direct use-case here.

3.5.2. Certified Constraint Programming

We plan to make use of the internalisation of the object-oriented paradigm in the context of constraint programming. Indeed, this domain is made of very complex algorithms that are often developed using object-oriented programming (as it is the case for instance for CHOCO, which is developed in the Tasc Group at IMT Atlantique, Nantes). We will in particular focus on filtering algorithms in constraint solvers, for which research publications currently propose new algorithms with manual proofs. Their formalisation in Coq is challenging. Another interesting part of constraint solving to formalise is the part that deals with program generation (as opposed to extraction). However, when there are numerous generated pieces of code, it is not realistic to prove their correctness manually, and it can be too difficult to prove the correctness of a generator. So we intend to explore a middle path that consists in generating a piece of code along with its corresponding proof (script or proof term). A target application could be interval constraints (for instance Allen interval algebra or region connection calculus) that can generate thousands of specialised filtering algorithms for a small number of variables [37].

Finally, Rémi Douence has already worked (articles publishing [64], [98], [54], PhD Thesis advising [99]) with different members of the Tasc team. Currently, he supervises with Nicolas Beldiceanu the PhD Thesis of Ekaterina Arafailova in the Tasc team. She studies finite transducers to model time-series constraints [38], [36], [35]. This work requires proofs, manually done for now, we would like to explore when these proofs could be mechanised.

3.5.3. Certified Symbolic Computation

We will investigate how the addition of effects in the Coq proof assistant can facilitate the marriage of computer algebra with formal proofs. Computer algebra systems on one hand, and proof assistants on the other hand, are both designed for doing mathematics with the help of a computer, by the means of symbolic computations. These two families of systems are however very different in nature: computer algebra systems allow for implementations faithful to the theoretical complexity of the algorithms, whereas proof assistants have the expressiveness to specify exactly the semantic of the data-structures and computations.

Experiments have been run that link computer algebra systems with Coq [53], [43]. These bridges rely on the implementation of formal proof-producing core algorithms like normalisation procedures. Incidentally, they require non trivial maintenance work to survive the evolution of both systems. Other proof assistants like the Isabelle/HOL system make use of so-called reflection schemes: the proof assistant can produce code in an external programming language like SML, but also allows to import the values output by these extracted programs back inside the formal proofs. This feature extends the trusted base of code quite significantly but it has been used for major achievements like a certified symbolic/numeric ODE solver [70].

We would like to bring Coq closer to the efficiency and user-friendliness of computer algebra systems: for now it is difficult to use the Coq programming language so that certified implementations of computer algebra algorithms have the right, observable, complexity when they are executed inside Coq. We see the addition of effects to the proof assistant as an opportunity to ease these implementations, for instance by making use of caching mechanisms or of profiling facilities. Such enhancements should enable the verification of computation-intensive mathematical proofs that are currently beyond reach, like the validation of Helfgott's proof of the weak Goldbach conjecture [66].

4. Highlights of the Year

4.1. Highlights of the Year

4.1.1. Creation and new permanents members

The team has been created as a project team on June 2018. Two new permanent members have joined the team:

- Guilhem Jaber, as an assistant professor of the University of Nantes (September 2018).
- Pierre-Marie Pédrot as an Inria researcher (October 2018).

4.1.2. Awards

BEST PAPERS AWARDS:

[7]
N. TABAREAU, É. TANTER, M. SOZEAU. *Equivalences for Free: Univalent Parametricity for Effective Transport*, in "Proceedings of the ACM on Programming Languages", September 2018, pp. 1-29 [*DOI :* 10.1145/3234615], https://hal.inria.fr/hal-01559073

[14]
É. MIQUEY. *A sequent calculus with dependent types for classical arithmetic*, in "LICS 2018 - 33th Annual ACM/IEEE Symposium on Logic in Computer Science", Oxford, United Kingdom, LICS '18 Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, ACM, July 2018, pp. 720-729, https://arxiv.org/abs/1805.09542 [*DOI :* 10.1145/3209108.3209199], https://hal.inria.fr/hal-01703526

5. New Software and Platforms

5.1. Coq

The Coq Proof Assistant
KEYWORDS: Proof - Certification - Formalisation

SCIENTIFIC DESCRIPTION: Coq is an interactive proof assistant based on the Calculus of (Co-)Inductive Constructions, extended with universe polymorphism. This type theory features inductive and co-inductive families, an impredicative sort and a hierarchy of predicative universes, making it a very expressive logic. The calculus allows to formalize both general mathematics and computer programs, ranging from theories of finite structures to abstract algebra and categories to programming language metatheory and compiler verification. Coq is organised as a (relatively small) kernel including efficient conversion tests on which are built a set of higher-level layers: a powerful proof engine and unification algorithm, various tactics/decision procedures, a transactional document model and, at the very top an IDE.
FUNCTIONAL DESCRIPTION: Coq provides both a dependently-typed functional programming language and a logical formalism, which, altogether, support the formalisation of mathematical theories and the specification and certification of properties of programs. Coq also provides a large and extensible set of automatic or semi-automatic proof methods. Coq's programs are extractible to OCaml, Haskell, Scheme, ...

RELEASE FUNCTIONAL DESCRIPTION: Coq version 8.8.2 contains the result of refinements and stabilization of features and deprecations, cleanups of the internals of the system along with a few new features.

Summary of changes:

Kernel: fix a subject reduction failure due to allowing fixpoints on non-recursive values (#407), by Matthieu Sozeau. Handling of evars in the VM (#935) by Pierre-Marie Pédrot.

Notations: many improvements on recursive notations and support for destructuring patterns in the syntax of notations by Hugo Herbelin.

Proof language: tacticals for profiling, timing and checking success or failure of tactics by Jason Gross. The focusing bracket { supports single-numbered goal selectors, e.g. 2:{, (#6551) by Théo Zimmermann.

Vernacular: cleanup of definition commands (#6653) by Vincent Laporte and more uniform handling of the Local flag (#1049), by Maxime Dénès. Experimental Show Extraction command (#6926) by Pierre Letouzey. Coercion now accepts Prop or Type as a source (#6480) by Arthur Charguéraud. Export modifier for options allowing to export the option to modules that Import and not only Require a module (#6923), by Pierre-Marie Pédrot.

Universes: many user-level and API level enhancements: qualified naming and printing, variance annotations for cumulative inductive types, more general constraints and enhancements of the minimization heuristics, interaction with modules by Gaëtan Gilbert, Pierre-Marie Pédrot and Matthieu Sozeau.

Library: Decimal Numbers library (#6599) by Pierre Letouzey and various small improvements.

Documentation: a large community effort resulted in the migration of the reference manual to the Sphinx documentation tool. The new documentation infrastructure (based on Sphinx) is by Clément Pit-Claudel. The migration was coordinated by Maxime Dénès and Paul Steckler, with some help of Théo Zimmermann during the final integration phase. The 14 people who ported the manual are Calvin Beck, Heiko Becker, Yves Bertot, Maxime Dénès, Richard Ford, Pierre Letouzey, Assia Mahboubi, Clément Pit-Claudel, Laurence Rideau, Matthieu Sozeau, Paul Steckler, Enrico Tassi, Laurent Théry, Nikita Zyuzin.

Tools: experimental -mangle-names option to coqtop/coqc for linting proof scripts (#6582), by Jasper Hugunin. Main changes:

Critical soundness bugs were fixed between versions 8.8.0 and 8.8.2, and a PDF version of the reference manual was made available. The Windows installer also includes many more external packages that can be individually selected for installation.

On the implementation side, the dev/doc/changes.md file documents the numerous changes to the implementation and improvements of interfaces. The file provides guidelines on porting a plugin to the new version.

More information can be found in the CHANGES file. Feedback and bug reports are extremely welcome.

Distribution Installers for Windows 32 bits (i686), Windows 64 bits (x8_64) and macOS are available. They come bundled with CoqIDE. Windows binaries now include the Bignums library.

Complete sources of the files installed by the Windows installers are made available, to comply with license requirements.
NEWS OF THE YEAR: Version 8.8.0 was released in April 2018 and version 8.8.2 in September 2018. This is the third release of Coq developed on a time-based development cycle. Its development spanned 6 months from the release of Coq 8.7 and was based on a public road-map. It attracted many external contributions. Code reviews and continuous integration testing were systematically used before integration of new features, with an important focus given to compatibility and performance issues.

The main advances in this version are cleanups and fixes in the many different components of the system, ranging from low level kernel fixes to advances in the support of notations and tacticals for selecting goals. A large community effort was made to move the documentation to the Sphinx format, providing a more accessible online ressource to users.

- Participants: Abhishek Anand, C. J. Bell, Yves Bertot, Frédéric Besson, Tej Chajed, Pierre Courtieu, Maxime Denes, Julien Forest, Emilio Jesús Gallego Arias, Gaëtan Gilbert, Benjamin Grégoire, Jason Gross, Hugo Herbelin, Ralf Jung, Matej Kosik, Sam Pablo Kuper, Xavier Leroy, Pierre Letouzey, Assia Mahboubi, Cyprien Mangin, Érik Martin-Dorel, Olivier Marty, Guillaume Melquiond, Pierre-Marie Pédrot, Benjamin C. Pierce, Lars Rasmusson, Yann Régis-Gianas, Lionel Rieg, Valentin Robert, Thomas Sibut-Pinote, Michael Soegtrop, Matthieu Sozeau, Arnaud Spiwack, Paul Steckler, George Stelle, Pierre-Yves Strub, Enrico Tassi, Hendrik Tews, Laurent Théry, Amin Timany, Vadim Zaliva and Théo Zimmermann
- Partners: CNRS - Université Paris-Sud - ENS Lyon - Université Paris-Diderot
- Contact: Matthieu Sozeau
- Publication: The Coq Proof Assistant, version 8.8.0
- URL: http://coq.inria.fr/

5.2. Math-Components

Mathematical Components library
FUNCTIONAL DESCRIPTION: The Mathematical Components library is a set of Coq libraries that cover the mechanization of the proof of the Odd Order Theorem.

RELEASE FUNCTIONAL DESCRIPTION: The library includes 16 more theory files, covering in particular field and Galois theory, advanced character theory, and a construction of algebraic numbers.

- Participants: Alexey Solovyev, Andrea Asperti, Assia Mahboubi, Cyril Cohen, Enrico Tassi, François Garillot, Georges Gonthier, Ioana Pasca, Jeremy Avigad, Laurence Rideau, Laurent Théry, Russell O'Connor, Sidi Ould Biha, Stéphane Le Roux and Yves Bertot
- Contact: Assia Mahboubi
- URL: http://math-comp.github.io/math-comp/

5.3. Ssreflect

FUNCTIONAL DESCRIPTION: Ssreflect is a tactic language extension to the Coq system, developed by the Mathematical Components team.

- Participants: Assia Mahboubi, Cyril Cohen, Enrico Tassi, Georges Gonthier, Laurence Rideau, Laurent Théry and Yves Bertot
- Contact: Yves Bertot
- URL: http://math-comp.github.io/math-comp/

5.4. Ltac2

KEYWORDS: Coq - Proof assistant

FUNCTIONAL DESCRIPTION: A replacement for Ltac, the tactic language of Coq.

- Contact: Pierre-Marie Pédrot

6. New Results

6.1. Logical Foundations of Programming Languages

Participants: Rémi Douence, Ambroise Lafont, Étienne Miquey, Xavier Montillet, Guillaume Munch-Maccagnoni, Nicolas Tabareau, Pierre Vial.

6.1.1. Classical Logic

6.1.1.1. A sequent calculus with dependent types for classical arithmetic.

In a recent paper, Herbelin developed a calculus $dPA\omega$ in which constructive proofs for the axioms of countable and dependent choices could be derived via the encoding of a proof of countable universal quantification as a stream of it components. However, the property of normalisation (and therefore the one of soundness) was only conjectured. The difficulty for the proof of normalisation is due to the simultaneous presence of dependent dependent types (for the constructive part of the choice), of control operators (for classical logic), of coinductive objects (to encode functions of type $N \to A$ into streams $(a_0, a_1, \cdots)$) and of lazy evaluation with sharing (for these coinductive objects).Building on previous works, we introduce in [14], [26] a variant of $dPA\omega$ presented as a sequent calculus. On the one hand, we take advantage of a variant of Krivine classical realisability we developed to prove the normalisation of classical call-by-need. On the other hand, we benefit of dL, a classical sequent calculus with dependent types in which type safety is ensured using delimited continuations together with a syntactic restriction. By combining the techniques developed in these papers, we manage to define a realisability interpretation à la Krivine of our calculus that allows us to prove normalisation and soundness.

6.1.1.2. Realisability Interpretation and Normalisation of Typed Call-by-Need λ-calculus With Control.

In [13], we define a variant of realisability where realisers are pairs of a term and a substitution. This variant allows us to prove the normalisation of a simply-typed call-by-need λ-calculus with control due to Ariola et al. Indeed, in such call-by-need calculus, substitutions have to be delayed until knowing if an argument is really needed. In a second step, we extend the proof to a call-by-need λ-calculus equipped with a type system equivalent to classical second-order predicate logic, representing one step towards proving the normalisation of the call-by-need classical second-order arithmetic introduced by the second author to provide a proof-as-program interpretation of the axiom of dependent choice.

6.1.2. Lambda Calculus

6.1.2.1. Every λ-Term is Meaningful for the Infinitary Relational Model.

Infinite types and formulas are known to have really curious and unsound behaviours. For instance, they allow to type Ω, the auto-autoapplication and they thus do not ensure any form of normalisation/productivity. Moreover, in most infinitary frameworks, it is not difficult to define a type R that can be assigned to every λ-term. However, these observations do not say much about what coinductive (i.e. infinitary) type grammars are able to provide: it is for instance very difficult to know what types (besides R) can be assigned to a given term in this setting. In [17], we begin with a discussion on the expressivity of different forms of infinite types. Then, using the resource-awareness of sequential intersection types (system S) and tracking, we prove that infinite types are able to characterise the arity of every λ-terms and that, in the infinitary extension of the relational model, every term has a " meaning " i.e. a non-empty denotation. From the technical point of view, we must deal with the total lack of guarantee of productivity for typable terms: we do so by importing methods inspired by first order model theory.

6.1.2.2. High-level signatures and initial semantics.

In [9], we present a device for specifying and reasoning about syntax for datatypes, programming languages, and logic calculi. More precisely, we consider a general notion of signature for specifying syntactic constructions. Our signatures subsume classical algebraic signatures (i.e., signatures for languages with variable binding, such as the pure lambda calculus) and extend to much more general examples. In the spirit of Initial Semantics, we define the syntax generated by a signature to be the initial object—if it exists—in a suitable category of models. Our notions of signature and syntax are suited for compositionality and provide, beyond the desired algebra of terms, a well-behaved substitution and the associated inductive/recursive principles. Our signatures are general in the sense that the existence of syntax is not automatically guaranteed. In this work, we identify a large class of signatures which do generate a syntax. This paper builds upon ideas from a previous attempt by Hirschowitz-Maggesi (FICS 2012), which, in turn, was directly inspired by some earlier work of Ghani-Uustalu and Matthes-Uustalu. The main results presented in the paper are computer-checked within the UniMath system.

6.1.3. Models of programming languages mixing effects and resources

6.1.3.1. A resource modality for RAII

Systems programming languages C++11 and Rust have developed techniques and idioms for the safe management of resources called *"Resource acquisition is initialisation"* (RAII) and *move semantics*. We have related resources from systems programming to the notion of resource put forward by linear logic, by giving a construction in terms of categorical semantics for a resource modality that model RAII and move semantics. This work was presented by at the workshop LOLA 2018 in Oxford [20].

6.1.3.2. Resource polymorphism

Thanks to a new logical and semantic understanding of resource-management techniques in systems programming languages, we have proposed [27] a design for an extension of functional programming language towards systems programming, centred on the OCaml language, and based on a notion of resource polymorphism inspired by the C++11 language and by the works on polarisation in proof theory.

6.1.4. Distributed Programming

6.1.4.1. Chemical foundations of distributed aspects.

Distributed applications are challenging to program because they have to deal with a plethora of concerns, including synchronisation, locality, replication, security and fault tolerance. Aspect-oriented programming (AOP) is a paradigm that promotes better modularity by providing means to encapsulate cross-cutting concerns in entities called aspects. Over the last years, a number of distributed aspect-oriented programming languages and systems have been proposed, illustrating the benefits of AOP in a distributed setting. Chemical calculi are particularly well-suited to formally specify the behaviour of concurrent and distributed systems. The join calculus is a functional name-passing calculus, with both distributed and object-oriented extensions. It is used as the basis of concurrency and distribution features in several mainstream languages like C# (Polyphonic C#, now Cω), OCaml (JoCaml), and Scala Joins. Unsurprisingly, practical programming in the join calculus also suffers from modularity issues when dealing with crosscutting concerns. We propose the Aspect Join Calculus [8], an aspect-oriented and distributed variant of the join calculus that addresses crosscutting and provides a formal foundation for distributed AOP. We develop a minimal aspect join calculus that allows aspects to advise chemical reactions. We show how to deal with causal relations in pointcuts and how to support advanced customisable aspect weaving semantics.

6.2. Type Theory and Proof Assistants

Participants: Simon Boulier, Eric Finster, Gaëtan Gilbert, Pierre-Marie Pédrot, Nicolas Tabareau, Théo Winterhalter.

6.2.1. Type Theory

6.2.1.1. Effects in Type Theory.

In [16] , we define the exceptional translation, a syntactic translation of the Calculus of Inductive Constructions (CIC) into itself, that covers full dependent elimination. The new resulting type theory features call-by-name exceptions with decidable type-checking and canonicity, but at the price of inconsistency. Then, noticing parametricity amounts to Kreisel's realisability in this setting, we provide an additional layer on top of the exceptional translation in order to tame exceptions and ensure that all exceptions used locally are caught, leading to the parametric exceptional translation which fully preserves consistency. This way, we can consistently extend the logical expressivity of CIC with independence of premises, Markov's rule, and the negation of function extensionality while retaining η-expansion. As a byproduct, we also show that Markov's principle is not provable in CIC. Both translations have been implemented in a Coq plugin, which we use to formalise the examples.

6.2.1.2. Eliminating Reflection from Type Theory.

Type theories with equality reflection, such as extensional type theory (ETT), are convenient theories in which to formalise mathematics, as they make it possible to consider provably equal terms as convertible. Although type-checking is undecidable in this context, variants of ETT have been implemented, for example in NuPRL and more recently in Andromeda. The actual objects that can be checked are not proof-terms, but derivations of proof-terms. This suggests that any derivation of ETT can be translated into a typecheckable proof term of intensional type theory (ITT). However, this result, investigated categorically by Hofmann in 1995, and 10 years later more syntactically by Oury, has never given rise to an effective translation. In [18], we provide the first syntactical translation from ETT to ITT with uniqueness of identity proofs and functional extensionality. This translation has been defined and proven correct in Coq and yields an executable plugin that translates a derivation in ETT into an actual Coq typing judgment. Additionally, we show how this result is extended in the context of homotopy to a two-level type theory.

6.2.1.3. Foundations of Dependent Interoperability.

Full-spectrum dependent types promise to enable the development of correct-by-construction software. However, even certified software needs to interact with simply-typed or untyped programs, be it to perform system calls, or to use legacy libraries. Trading static guarantees for runtime checks, the dependent interoperability framework provides a mechanism by which simply-typed values can safely be coerced to dependent types and, conversely, dependently-typed programs can defensively be exported to a simply-typed application. In [2], we give a semantic account of dependent interoperability. Our presentation relies on and is guided by a pervading notion of type equivalence, whose importance has been emphasised in recent work on homotopy type theory. Specifically, we develop the notion of type-theoretic partial Galois connections as a key foundation for dependent interoperability, which accounts for the partiality of the coercions between types. We explore the applicability of both monotone and antitone type-theoretic Galois connections in the setting of dependent interoperability. A monotone partial Galois connection enforces a translation of dependent types to runtime checks that are both sound and complete with respect to the invariants encoded by dependent types. Conversely, picking an antitone partial Galois connection instead lets us induce weaker, sound conditions that can amount to more efficient runtime checks. Our framework is developed in Coq; it is thus constructive and verified in the strictest sense of the terms. Using our library, users can specify domain-specific partial connections between data structures. Our library then takes care of the (sometimes, heavy) lifting that leads to interoperable programs. It thus becomes possible, as we shall illustrate, to internalise and hand-tune the extraction of dependently-typed programs to interoperable OCaml programs within Coq itself.

6.2.1.4. Equivalences for Free: Univalent Parametricity for Effective Transport.

Homotopy Type Theory promises a unification of the concepts of equality and equivalence in Type Theory, through the introduction of the univalence principle. However, existing proof assistants based on type theory treat this principle as an axiom, and it is not yet clear how to extend them to handle univalence internally. In [7], we propose a construction grounded on a univalent version of parametricity to bring the benefits of univalence to the programmer and prover, that can be used on top of existing type theories. In particular, univalent

parametricity strengthens parametricity to ensure preservation of type equivalences. We present a lightweight framework implemented in the Coq proof assistant that allows the user to transparently transfer definitions and theorems for a type to an equivalent one, as if they were equal. Our approach handles both type and term dependency. We study how to maximise the effectiveness of these transports in terms of computational behaviour, and identify a fragment useful for certified programming on which univalent transport is guaranteed to be effective. This work paves the way to easier-to-use environments for certified programming by supporting seamless programming and proving modulo equivalences.

6.2.1.5. Special Issue on Homotopy Type Theory and Univalent Foundations.

The preface [4] introduces the first special issue out of a series of workshops on Homotopy Type Theory and Univalent Foundations. This recent area of research finds its roots in the seminal work of Martin Hofmann and Thomas Streicher on the structure of Martin-Löf identity types. But the main research program has been foreseen by Vladimir Voevodsky, who, from its initial motivation of formalising his results in homotopy theory, has initiated what is now called the univalent foundations program. Borrowing ideas from homotopy theory, the goal of the univalent foundations program is to leverage dependent Type Theory to a formal framework that could replace Set Theory for the foundations of mathematics. This special issue gathers research contributions of some of the most prominent researchers of the field.

6.2.1.6. Goodwillie's Calculus of Functors and Higher Topos Theory

In [1], we develop an approach to Goodwillie's calculus of functors using the techniques of higher topos theory. Central to our method is the introduction of the notion of fiberwise orthogonality, a strengthening of ordinary orthogonality which allows us to give a number of useful characterisations of the class of n-excisive maps. We use these results to show that the pushout product of a P_n-equivalence with a P_m-equivalence is a P_{m+n+1}-equivalence. Then, building on our previous work, we prove a Blakers-Massey type theorem for the Goodwillie tower. We show how to use the resulting techniques to rederive some foundational theorems in the subject, such as delooping of homogeneous functors.

6.2.2. Proof Assistants

6.2.2.1. Typed Template Coq – Certified Meta-Programming in Coq.

Template-Coq [19], [10] is a plugin for Coq, originally implemented by Malecha, which provides a reifier for Coq terms and global declarations , as represented in the Coq kernel, as well as a denotation command. Initially, it was developed for the purpose of writing functions on Coq's AST in Gallina. Recently, it was used in the CertiCoq certified compiler project, as its front-end language, to derive parametricity properties, and to extract Coq terms to a CBV λ-calculus. However, the syntax lacked semantics, be it typing semantics or operational semantics, which should reflect, as formal specifications in Coq, the semantics of Coq's type theory itself. The tool was also rather bare bones, providing only rudimentary quoting and unquoting commands. We generalise it to handle the entire Calculus of Inductive Constructions (CIC), as implemented by Coq, including the kernel's declaration structures for definitions and inductives, and implement a monad for general manipulation of Coq's logical environment. We demonstrate how this setup allows Coq users to define many kinds of general purpose plugins, whose correctness can be readily proved in the system itself, and that can be run efficiently after extraction. We give a few examples of implemented plugins, including a parametricity translation. We also advocate the use of Template-Coq as a foundation for higher-level tools.

6.2.2.2. Definitional Proof-Irrelevance without K.

Definitional equality—or conversion—for a type theory with a decidable type checking is the simplest tool to prove that two objects are the same, letting the system decide just using computation. Therefore, the more things are equal by conversion, the simpler it is to use a language based on type theory. Proof-irrelevance, stating that any two proofs of the same proposition are equal, is a possible way to extend conversion to make a type theory more powerful. However, this new power comes at a price if we integrate it naively, either by making type checking undecidable or by realising new axioms—such as uniqueness of identity proofs (UIP)—that are incompatible with other extensions, such as univalence. In [3], taking inspiration from homotopy type theory, we propose a general way to extend a type theory with definitional proof irrelevance, in a way that keeps type checking decidable and is compatible with univalence. We provide a new criterion to

decide whether a proposition can be eliminated over a type (correcting and improving the so-called singleton elimination of Coq) by using techniques coming from recent development on dependent pattern matching without UIP. We show the generality of our approach by providing implementations for both Coq and Agda, both of which are planned to be integrated in future versions of those proof assistants.

6.3. Program Certifications and Formalisation of Mathematics

Participants: Danil Annenkov, Assia Mahboubi, Étienne Miquey.

6.3.1. Certified Compilation of Financial Contracts.

In [11], we present an extension to a certified financial contract management system that allows for templated declarative financial contracts and for integration with financial stochastic models through verified compilation into so-called payoff-expressions. Such expressions readily allow for determining the value of a contract in a given evaluation context, such as contexts created for stochastic simulations. The templating mechanism is useful both at the contract specification level, for writing generic reusable contracts, and for reuse of code that, without the templating mechanism, needs to be recompiled for different evaluation contexts. We report on the effect of using the certified system in the context of a GPGPU-based Monte Carlo simulation engine for pricing various over-the-counter (OTC) financial contracts. The full contract-management system, including the payoff-language compilation, is verified in the Coq proof assistant and certified Haskell code is extracted from our Coq development along with Futhark code for use in a data-parallel pricing engine.

6.3.2. Static interpretation of higher-order modules in Futhark: functional GPU programming in the large.

In [12], we present a higher-order module system for the purely functional data-parallel array language Futhark. The module language has the property that it is completely eliminated at compile time, yet it serves as a powerful tool for organising libraries and complete programs. The presentation includes a static and a dynamic semantics for the language in terms of, respectively, a static type system and a provably terminating elaboration of terms into terms of an underlying target language. The development is formalised in Coq using a novel encoding of semantic objects based on products, sets, and finite maps. The module language features a unified treatment of module type abstraction and core language polymorphism and is rich enough for expressing practical forms of module composition.

6.3.3. Formalising Implicative Algebras in Coq.

In [15], we present a Coq formalisation of Alexandre Miquel's implicative algebras, which aim at providing a general algebraic framework for the study of classical realisability models. We first give a self-contained presentation of the underlying implicative structures, which roughly consists of a complete lattice equipped with a binary law representing the implication. We then explain how these structures can beturned into models by adding separators, giving rise to the so-called implicative algebras. Additionally, we show how they generalise Boolean and Heyting algebras as well as the usual algebraic structures used in the analysis of classical realisability.

6.3.4. Formally Verified Approximations of Definite Integrals.

Finding an elementary form for an antiderivative is often a difficult task, so numerical integration has become a common tool when it comes to making sense of a definite integral. Some of the numerical integration methods can even be made rigorous: not only do they compute an approximation of the integral value but they also bound its inaccuracy. Yet numerical integration is still missing from the toolbox when performing formal proofs in analysis. In [5], we present an efficient method for automatically computing and proving bounds on some definite integrals inside the Coq formal system. Our approach is not based on traditional quadrature methods such as Newton-Cotes formulas. Instead, it relies on computing and evaluating antiderivatives of rigorous polynomial approximations, combined with an adaptive domain splitting. Our approach also handles improper integrals, provided that a factor of the integrand belongs to a catalog of identified integrable functions. This work has been integrated to the CoqInterval library.

7. Partnerships and Cooperations

7.1. Regional Initiatives

Vercoma (Atlanstic 2020/Attractivity grant)

Goal: Verified comupter mathematics.

Coordinator: A. Mahboubi.

Duration: 08/2018 - 08/2021.

7.2. National Initiatives

7.2.1. ANR

FastRelax (ANR-14-CE25-0018).

Goal: Develop computer-aided proofs of numerical values, with certified and reasonably tight error bounds, without sacrificing efficiency.

Coordinator: Bruno Salvy (Inria, ENS Lyon).

Participant: A. Mahboubi.

Duration: 2014-2019.

Website: http://fastrelax.gforge.inria.fr/.

Note: This project started when A. Mahboubi was still in the Specfun project at the Saclay Île-de-France CRI. The budget is still managed there, within the Toccata project, but remains available to A. Mahboubi.

7.3. European Initiatives

7.3.1. H2020 Projects

7.3.1.1. CoqHoTT

Title: Coq for Homotopy Type Theory

Programm: H2020

Type: ERC

Duration: June 2015 - May 2020

Coordinator: Inria

Inria contact: Nicolas TABAREAU

Every year, software bugs cost hundreds of millions of euros to companies and administrations. Hence, software quality is a prevalent notion and interactive theorem provers based on type theory have shown their efficiency to prove correctness of important pieces of software like the C compiler of the CompCert project. One main interest of such theorem provers is the ability to extract directly the code from the proof. Unfortunately, their democratization suffers from a major drawback, the mismatch between equality in mathematics and in type theory. Thus, significant Coq developments have only been done by virtuosos playing with advanced concepts of computer science and mathematics. Recently, an extension of type theory with homotopical concepts such as univalence is gaining traction because it allows for the first time to marry together expected principles of equality. But the univalence principle has been treated so far as a new axiom which breaks one fundamental property of mechanized proofs: the ability to compute with programs that make use of this axiom. The main goal of the CoqHoTT project is to provide a new generation of proof assistants with a computational version of univalence and use them as a base to implement effective logical model transformation so that the power of the internal logic of the proof assistant needed

to prove the correctness of a program can be decided and changed at compile time—according to a trade-off between efficiency and logical expressivity. Our approach is based on a radically new compilation phase technique into a core type theory to modularize the difficulty of finding a decidable type checking algorithm for homotopy type theory. The impact of the CoqHoTT project will be very strong. Even if Coq is already a success, this project will promote it as a major proof assistant, for both computer scientists and mathematicians. CoqHoTT will become an essential tool for program certification and formalization of mathematics.

7.3.2. Collaborations in European Programs, Except FP7 & H2020

Program: COST

Project acronym: EUTYPES

Project title: The European research network on types for programming and verification

Duration: 21/03/2016 - 20/03/2020.

Coordinator: Herman Geuvers (Radboud Univerity, Nijmegen, The Netherlands)

Abstract: Types are pervasive in programming and information technology. A type defines a formal interface between software components, allowing the automatic verification of their connections, and greatly enhancing the robustness and reliability of computations and communications. In rich dependent type theories, the full functional specification of a program can be expressed as a type. Type systems have rapidly evolved over the past years, becoming more sophisticated, capturing new aspects of the behaviour of programs and the dynamics of their execution.

This COST Action will give a strong impetus to research on type theory and its many applications in computer science, by promoting (1) the synergy between theoretical computer scientists, logicians and mathematicians to develop new foundations for type theory, for example as based on the recent development of "homotopy type theory", (2) the joint development of type theoretic tools as proof assistants and integrated programming environments, (3) the study of dependent types for programming and its deployment in software development, (4) the study of dependent types for verification and its deployment in software analysis and verification. The action will also tie together these different areas and promote cross-fertilisation.

Europe has a strong type theory community, ranging from foundational research to applications in programming languages, verification and theorem proving, which is in urgent need of better networking. A COST Action that crosses the borders will support the collaboration between groups and complementary expertise, and mobilise a critical mass of existing type theory research.

7.4. International Initiatives

7.4.1. Inria International Labs

Inria Chile

Associate Team involved in the International Lab:

7.4.1.1. GECO

Title: Gradual verification and robust proof Engineering for COq

International Partner (Institution - Laboratory - Researcher):

Universidad de Chile (Chile) - DCC, Pleaid team - Éric Tanter

Start year: 2018

See also: http://geco.gforge.inria.fr

The development of tools to construct software systems that respect a given specification is a major challenge of current and future research in computer science. Interactive theorem provers based on type theory, such as Coq, have shown their effectiveness to prove correctness of important pieces of software like the C compiler of the CompCert project. Certified programming with dependent types is attracting a lot of attention recently, and Coq is the de facto standard for such endeavors, with an increasing amount of users, pedagogical material, and large-scale projects. Nevertheless, significant work remains to be done to make Coq more usable from a software engineering point of view.

This collaboration project gathers the expertise of researchers from Chile (Inria Chile, Universidad de Chile, Universidad Católica de Valparaíso) and France (Inria Nantes, Inria Paris), in different areas that are crucial to develop the vision of certified software engineering. The focus of this project is both theoretical and practical, covering novel foundations and methods, design of concrete languages and tools, and validation through specific case studies.

The end result will be a number of enhancements to the Coq proof assistant (frameworks, tactic language) together with guidelines and demonstrations of their applicability in realistic scenarios.

7.5. International Research Visitors

7.5.1. Visits of International Scientists

- Ambrus Kaposi has visited Gallinette from April 15 to July 15 as part of the ERC 'Visiting Fellowship Programmes' whose aims it to promote the widening of participation of researchers with a high potential in the ERC calls. The Scientific Council of the ERC believes that increasing the international exposure of researchers can help them to develop their full research potential. For this reason the ERC has invited relevant national and regional authorities in Europe to fund potential ERC candidates from the country or the region to visit teams of existing ERC Principal Investigators. The purpose is to offer these potential candidates an opportunity to broaden and strengthen their research profile and vision in an internationally competitive research environment before applying for an ERC grant.
- Simon Huber (University of Göteborg) has visited Simon Boulier and Nicolas Tabareau from Feb 26 to March 2 as a Short-Term Scientific Mission (STSM) funded by the EUTYPES COST Action.
- Jesper Cockx (Chalmers University) has visited Gaetan Gilbert and Nicolas Tabareau from Feb 19 to Feb 23 as a Short-Term Scientific Mission (STSM) funded by the EUTYPES COST Action.

7.5.1.1. Internships

- A. Defourné has visited the team from April to August for an internship on the subject "A Mini-ML with resource management", supervised by G. Munch-Maccagnoni and R. Douence.
- L. Pujet has visited the team from April to August for an internship on the subject "Interpreting Cubical Type Theory using forcing", supervised by N. Tabareau.

7.5.2. Visits to International Teams

7.5.2.1. Research Stays Abroad

- A. Mahboubi has been appointed as Endowed Professor at the Vrije Universiteit Amsterdam on a chair entitled "Automated verification of mathematical proof".
- G. Munch-Maccagnoni has visited the University of Cambridge from July 9th to July 19th to work with M. Fiore on the topic of categorical semantics of effects and resources in programming languages.
- G. Munch-Maccagnoni has visited Jane Street on July 19th to work with L. White on the topic of resource management in the OCaml programming language.

8. Dissemination

8.1. Promoting Scientific Activities

8.1.1. Scientific Events Organisation

8.1.1.1. General Chair, Scientific Chair

- N. Tabareau has co-organized (with Matthieu Sozeau) the Coq Workshop as part of FLoC 2018.
- A. Mahboubi has co-chaired (with Jeremy Avigad) the ITP 2018 conference, part of FLoC 2018.

8.1.1.2. Member of the Organizing Committees

- A. Mahboubi has co-organized (with Andrej Bauer, Martin Escardó and Peter Le Fanu Lumsdaine) the Dagsthul seminar 19341, Formalization of Mathematics in Type Theory.

8.1.2. Scientific Events Selection

8.1.2.1. Chair of Conference Program Committees

- A. Mahboubi has served in the FLoC 2018 Program Committee and Workshops Committee.

8.1.2.2. Member of the Conference Program Committees

- N. Tabareau has been a member of the program committee of FSCD'18 and ICFP'18 (External Review Committee).
- A. Mahboubi has been a member of the program committee of the CPP 2018 and CICM 2018 conferences, and of the TYPES 2018 and HOTT/UF 2018 workshops.

8.1.2.3. Reviewer

- N. Tabareau has served as an external reviewer for MFCS'18, WoLLIC'18.
- A. Mahboubi has served as external reviewer for the post-proceedings of TYPES 2017.

8.1.3. Journal

8.1.3.1. Member of the Editorial Boards

- A. Mahboubi is a member of the editorial board of the Journal of Automated Reasoning.
- A. Mahboubi is associate editor of the Progress in Computer Science and Applied Logic series.

8.1.3.2. Reviewer - Reviewing Activities

- A. Mahboubi has served as reviewer for the Journal of Automated Reasoning, and for the Annals of Mathematics and Artifical Intelligence.
- G. Munch-Maccagnoni has served as reviewer for the Journal of Automated Reasoning, and for Logical Methods in Computer Science.

8.1.4. Invited Talks

- A. Mahboubi has given an invited talk at the conference in honour of Thomas C. Hales on the occasion of his 60th birthday.
- A. Mahboubi has given an invited course at the Journées Nationales du Calcul Formel 2018.
- A. Mahboubi has given an invited course at the 19th Journées Louis Antoine.
- A. Mahboubi and G. Munch-Maccagnoni have given invited talks in the seminar organized by Xavier Leroy at the Collège de France in December.
- G. Munch-Maccagnoni has given an invited talk at the Journées Inaugurales du GT Scalp in November.
- G. Munch-Maccagnoni has given an invited talk at the seminar of the Celtique team in Rennes in October.
- G. Munch-Maccagnoni has given an invited talk at the seminar of the Gallium team in Paris in June.

- G. Munch-Maccagnoni has given an invited talk at the Logic and Semantics seminar at the University of Cambridge in June.

8.1.5. Leadership within the Scientific Community

- A. Mahboubi is a member of the scientific committee of the GdR Informatique-Mathématiques.
- A. Mahboubi is MC member for the COST Action CA15123 EUTypes, and a member of the core managment group of the project. She is leading the working group "Type-Theoretic Tools".

8.1.6. Research Administration

- A. Mahboubi has served in the Inria commitee examining the "Candidatures en détachements".
- A. Mahboubi has served as internal examiner in the jury of two "Maître de conférence" positions at Université de Nantes.
- A. Mahboubi has served in the jury of an assistant professor position in computer science at Stockholm University (Sweden).

8.2. Teaching - Supervision - Juries

8.2.1. Teaching

Licence : Julien Cohen, Discrete Mathematics, 48h, L1 (IUT Informatique), IUT Nantes, France

Licence : Object oriented programming, 32h, L3 (Ingineering school), Polytech Nantes, France

Master : Functional programming, 18h, M1 (Ingineering school), Polytech Nantes, France

Master : Tools for softaware engineering (proof, test, code management), 20h, M1 (Ingineering school), Polytech Nantes, France

Licence : Rémi Douence, Object Oriented Programming Project, 45h, L1 (apprenticeship), IMT-Atlantique, Nantes, France

Licence : Rémi Douence, Object Oriented Design and Programming, 25h , L3 (engineers), IMT-Atlantique, Nantes, France

Licence : Hervé Grall, Object Oriented Design and Programming, 25h , L3 (engineers), IMT-Atlantique, Nantes, France

Licence : Rémi Douence, Data Structures and Algorithms, 20h , L2 (engineers), IMT-Atlantique, Nantes, France

Licence : Rémi Douence, GUI, 20h, L2 (engineers), IMT-Atlantique, Nantes, France

Licence, Master : Hervé Grall, Modularity and Typing, 40h, L3 and M1, IMT-Atlantique, Nantes, France

Licence : Guilhem Jaber, Computer Tools for Science, 36h, L1, Université de Nantes France

Licence : Guilhem Jaber, Computer Architecure, 36h, L3, Université de Nantes France

Master : Hervé Grall, Service-oriented Computing, 40h, M1 and M2, IMT-Atlantique, Nantes, France

Master : Rémi Douence, Functional Programming with Haskell, 20h, M1 (engineers), IMT-Atlantique, Nantes, France

Master : Rémi Douence, Functional Programming with Haskell, 20h, M1 (apprenticeship), IMT-Atlantique, Nantes, France

Master : Rémi Douence, Introduction to scientific research in computer science (Project: an Haskell interpreter in Java), 45h, M2 (apprenticeship), IMT-Atlantique, Nantes, France

Master : Hervé Grall, Research Project - Certified Programming in Coq, 90h (1/3 supervised), M1 and M2, IMT-Atlantique, Nantes, France

Master : Nicolas Tabareau, Homotopy Type Theory, 24h, M2 LMFI, Université Paris Diderot, France

Master : Guilhem Jaber, Verification and Formal Proofs, 18h, M1, Université de Nantes, France

8.2.2. Supervision

PhD : Simon Boulier, Extending Type Theory with Syntactical Models, IMT Altantique, 29 Nov 2018, advisor: Nicolas Tabareau

PhD in progress : Antoine Allioux, Coherent Structures in Dependent Type Theory and Higher Category Theory, Université Paris Diderot, advisors: Yves Guiraud and Matthieu Sozeau

PhD in progress : Gaetan Gilbert, A new foundation for the Coq proof assistant based on the insight of Homotopy Type Theory, IMT Atlantique, advisors: Matthieu Sozeau and Nicolas Tabareau

PhD in progress : Ambroise Lafont,Towards an unbiased approach to specify, implement, and prove properties on programming languages, IMT Atlantique, advisors: Tom Hirschowitz and Nicolas Tabareau

PhD in progress: Xavier Montillet, Rewriting theory for effects and dependent types, Univ Nantes, advisors: Guillaume Munch-Maccagnoni and Nicolas Tabareau

PhD in progress: Théo Winterhalter, Extending the flexibility of the universe hierarchy in type theory, Univ Nantes, advisors: Matthieu Sozeau and Nicolas Tabareau

PhD in progress: Igor Zhirkov, Certified Refactoring of C in the Coq proof assistant, advisors: Rémi Douence and Julien Cohen.

PhD in progress: Joachim Hotonnier, Deep Specification for Domain-Specific Modelling, advisors: Gerson Sunye (Naomod team), Massimo Tisi (Naomod team), Hervé Grall

8.2.3. Juries

- A. Mahboubi has served as reviewer for the PhD of Andrea Gabrielli, defended October 22nd at Università di Firenze.
- A. Mahboubi has served as external member on the PhD jury of Pierre Boutry, defended November 13th at Université de Strasbourg.
- A. Mahboubi has served as external member on the PhD jury of Simon Boulier, defended November 29th at IMT Atlantique.
- A. Mahboubi has served as external member on the PhD jury of Boris Djalal, defended December 3rd at IMT Atlantique.
- J. Cohen has served as external member on the PhD jury of Thibaut Girka, defended July 3rd at Université Paris Diderot.

8.3. Popularization

8.3.1. Interventions

- A. Mahboubi is working with composer Alessandro Bossetti and students of the professional high-school Lycée Michelet, on a project “Art and Mathematics”, supported by the Athenor theater.

9. Bibliography

Publications of the year

Articles in International Peer-Reviewed Journals

[1] M. ANEL, G. BIEDERMANN, E. FINSTER, A. JOYAL. *Goodwillie’s calculus of functors and higher topos theory*, in "Journal of topology", December 2018, vol. 11, n^{o} 4, pp. 1100-1132 [*DOI :* 10.1112/TOPO.12082], https://hal.inria.fr/hal-01939906

[2] P.-E. DAGAND, N. TABAREAU, É. TANTER. *Foundations of Dependent Interoperability*, in "Journal of Functional Programming", March 2018, vol. 28 [*DOI :* 10.1017/S0956796818000011], https://hal.inria.fr/hal-01629909

[3] G. GILBERT, J. COCKX, M. SOZEAU, N. TABAREAU. *Definitional Proof-Irrelevance without K*, in "Proceedings of the ACM on Programming Languages", January 2019, https://hal.inria.fr/hal-01859964

[4] P. LEFANU LUMSDAINE, N. TABAREAU. *Preface: Special Issue on Homotopy Type Theory and Univalent Foundations*, in "Journal of Automated Reasoning", October 2018 [*DOI :* 10.1007/s10817-018-9491-3], https://hal.inria.fr/hal-01929871

[5] A. MAHBOUBI, G. MELQUIOND, T. SIBUT-PINOTE. *Formally Verified Approximations of Definite Integrals*, in "Journal of Automated Reasoning", March 2018, pp. 1-20 [*DOI :* 10.1007/s10817-018-9463-7], https://hal.inria.fr/hal-01630143

[6] É. MIQUEY. *A Classical Sequent Calculus with Dependent Types*, in "ACM Transactions on Programming Languages and Systems (TOPLAS)", 2018, https://hal.inria.fr/hal-01519929

[7] *Best Paper*
N. TABAREAU, É. TANTER, M. SOZEAU. *Equivalences for Free: Univalent Parametricity for Effective Transport*, in "Proceedings of the ACM on Programming Languages", September 2018, pp. 1-29 [*DOI :* 10.1145/3234615], https://hal.inria.fr/hal-01559073.

[8] N. TABAREAU, É. TANTER. *Chemical foundations of distributed aspects*, in "Distributed Computing", June 2018, pp. 1-24 [*DOI :* 10.1007/s00446-018-0334-6], https://hal.inria.fr/hal-01811884

International Conferences with Proceedings

[9] B. AHRENS, A. HIRSCHOWITZ, A. LAFONT, M. MAGGESI. *High-level signatures and initial semantics*, in "27th EACSL Annual Conference on Computer Science Logic (CSL 2018)", Birmingham, United Kingdom, September 2018, pp. 1-20, https://arxiv.org/abs/1805.03740 [*DOI :* 10.4230/LIPIcs.CSL.2018.4], https://hal.inria.fr/hal-01930058

[10] A. ANAND, S. BOULIER, C. COHEN, M. SOZEAU, N. TABAREAU. *Towards Certified Meta-Programming with Typed Template-Coq*, in "ITP 2018 - 9th Conference on Interactive Theorem Proving", Oxford, United Kingdom, LNCS, Springer, July 2018, vol. 10895, pp. 20-39 [*DOI :* 10.1007/978-3-319-94821-8_2], https://hal.archives-ouvertes.fr/hal-01809681

[11] D. ANNENKOV, M. ELSMAN. *Certified Compilation of Financial Contracts*, in "PPDP '18 - 20th International Symposium on Principles and Practice of Declarative Programming", Frankfurt am Main, Germany, ACM Press, September 2018, pp. 1-13 [*DOI :* 10.1145/3236950.3236955], https://hal.inria.fr/hal-01883559

[12] M. ELSMAN, T. HENRIKSEN, D. ANNENKOV, C. OANCEA. *Static interpretation of higher-order modules in Futhark: functional GPU programming in the large*, in "ICFP 2018 - International Conference on Functional Programming", St. Louis, United States, ACM, September 2018, vol. 2, n^{o} ICFP, pp. 1-30 [*DOI :* 10.1145/3236792], https://hal.inria.fr/hal-01883524

[13] É. MIQUEY, H. HERBELIN. *Realizability Interpretation and Normalization of Typed Call-by-Need λ-calculus With Control*, in "FOSSACS 18 - 21st International Conference on Foundations of Software Science and Computation Structures", Thessalonique, Greece, C. BAIER, U. D. LAGO (editors), LNCS, Springer, April 2018, vol. 10803, pp. 276-292, https://arxiv.org/abs/1803.00914 [*DOI :* 10.1007/978-3-319-89366-2_15], https://hal.inria.fr/hal-01624839

[14] *Best Paper*
É. MIQUEY. *A sequent calculus with dependent types for classical arithmetic*, in "LICS 2018 - 33th Annual ACM/IEEE Symposium on Logic in Computer Science", Oxford, United Kingdom, LICS '18 Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, ACM, July 2018, pp. 720-729, https://arxiv.org/abs/1805.09542 [*DOI :* 10.1145/3209108.3209199], https://hal.inria.fr/hal-01703526.

[15] É. MIQUEY. *Formalizing Implicative Algebras in Coq*, in "ITP 2018 - 9th International Conference on Interactive Theorem Proving", Oxford, United Kingdom, Lecture Notes in Computer Science, Springer, July 2018, vol. 10895, pp. 459-476 [*DOI :* 10.1007/978-3-319-94821-8_27], https://hal.inria.fr/hal-01703524

[16] P.-M. PÉDROT, N. TABAREAU. *Failure is Not an Option An Exceptional Type Theory*, in "ESOP 2018 - 27th European Symposium on Programming", Thessaloniki, Greece, LNCS, Springer, April 2018, vol. 10801, pp. 245-271 [*DOI :* 10.1007/978-3-319-89884-1_9], https://hal.inria.fr/hal-01840643

[17] P. VIAL. *Every λ-Term is Meaningful for the Infinitary Relational Model*, in "LICS 2018 - Thirty-Third Annual ACM/IEEE Symposium on Logic in Computer Science", Oxford, United Kingdom, E. G. ANUJ DAWAR (editor), Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018, ACM, July 2018, pp. 1-26 [*DOI :* 10.1145/NNNNNNN.NNNNNNN], https://hal.archives-ouvertes.fr/hal-01840744

[18] T. WINTERHALTER, M. SOZEAU, N. TABAREAU. *Eliminating Reflection from Type Theory: To the Legacy of Martin Hofmann*, in "CPP 2019 - The 8th ACM SIGPLAN International Conference on Certified Programs and Proofs", Lisbonne, Portugal, January 2019, pp. 1-19 [*DOI :* 10.1145/NNNNNNN.NNNNNNN], https://hal.archives-ouvertes.fr/hal-01849166

Conferences without Proceedings

[19] A. ANAND, S. BOULIER, N. TABAREAU, M. SOZEAU. *Typed Template Coq – Certified Meta-Programming in Coq*, in "CoqPL 2018 - The Fourth International Workshop on Coq for Programming Languages", Los Angeles, CA, United States, January 2018, pp. 1-2, https://hal.inria.fr/hal-01671948

[20] G. COMBETTE, G. MUNCH-MACCAGNONI. *A resource modality for RAII*, in "LOLA 2018: Workshop on Syntax and Semantics of Low-Level Languages", Oxford, United Kingdom, July 2018, pp. 1-4, https://hal.inria.fr/hal-01806634

Research Reports

[21] E. ARAFAILOVA, N. BELDICEANU, R. DOUENCE, M. CARLSSON, P. FLENER, J. PEARSON, M. A. FRANCISCO RODRÍGUEZ, H. SIMONIS. *Global Constraint Catalog, Volume II, Time-Series Constraints*, IMT Atlantique, September 2018, pp. 1-3762, https://arxiv.org/abs/1609.08925v2 , https://hal.inria.fr/hal-01374721

[22] W. BENGHABRIT, H. GRALL, J.-C. ROYER, A. SANTANA DE OLIVEIRA. *The Abstract Accountability Language: its Syntax, Semantics and Tools*, IMT Atlantique, August 2018, https://hal.archives-ouvertes.fr/hal-01856329

Other Publications

[23] T. COQ DEVELOPMENT TEAM. *The Coq Proof Assistant, version 8.8.0*, April 2018, Software [*DOI :* 10.5281/ZENODO.1219885], https://hal.inria.fr/hal-01954564

[24] H. HERBELIN, É. MIQUEY. *Continuation-and-environment-passing style translations: a focus on call-by-need*, January 2019, working paper or preprint, https://hal.inria.fr/hal-01972846

[25] M. KERJEAN, J.-S. LEMAY. *Higher-order distributions for differential linear logic.*, January 2019, working paper or preprint, https://hal.inria.fr/hal-01969262

[26] É. MIQUEY. *A Classical Sequent Calculus with Dependent Types*, March 2018, working paper or preprint, https://hal.archives-ouvertes.fr/hal-01744359

[27] G. MUNCH-MACCAGNONI. *Resource Polymorphism*, March 2018, working paper or preprint, https://hal.inria.fr/hal-01724997

References in notes

[28] A. ABEL, T. COQUAND. *Untyped Algorithmic Equality for Martin-Löf's Logical Framework with Surjective Pairs*, in "Typed Lambda Calculi and Applications", P. URZYCZYN (editor), Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2005, vol. 3461, pp. 23-38, http://dx.doi.org/10.1007/11417170_4

[29] B. ACCATTOLI, G. GUERRIERI. *Open Call-by-Value*, in "Programming Languages and Systems", January 2016, http://dx.doi.org/10.1007/978-3-319-47958-3_12

[30] D. AHMAN, N. GHANI, G. D. PLOTKIN. *Dependent Types and Fibred Computational Effects*, in "Proc. FoSSaCS", 2015

[31] A. AJOULI, J. COHEN, J.-C. ROYER. *Transformations between Composite and Visitor Implementations in Java*, in "Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on", Sept 2013, pp. 25–32, http://dx.doi.org/10.1109/SEAA.2013.53

[32] J. ALDRICH. *The power of interoperability: why objects are inevitable*, in "ACM Symposium on New Ideas in Programming and Reflections on Software, Onward! 2013, part of SPLASH '13, Indianapolis, IN, USA, October 26-31, 2013", A. L. HOSKING, P. T. EUGSTER, R. HIRSCHFELD (editors), ACM, 2013, pp. 101–116

[33] T. ALTENKIRCH, C. MCBRIDE, W. SWIERSTRA. *Observational equality, now!*, in "Proceedings of the ACM Workshop on Programming Languages meets Program Verification (PLPV 2007)", Freiburg, Germany, October 2007, pp. 57–68

[34] J.-M. ANDREOLI. *Logic Programming with Focusing Proof in Linear Logic*, in "Journal of Logic and Computation", 1992, vol. 2, n^{o} 3, pp. 297-347

[35] E. ARAFAILOVA, N. BELDICEANU, R. DOUENCE, M. CARLSSON, P. FLENER, M. A. F. RODRÍGUEZ, J. PEARSON, H. SIMONIS. *Global Constraint Catalog, Volume II, Time-Series Constraints*, in "CoRR", 2016, vol. abs/1609.08925, http://arxiv.org/abs/1609.08925

[36] E. ARAFAILOVA, N. BELDICEANU, R. DOUENCE, P. FLENER, M. A. F. RODRÍGUEZ, J. PEARSON, H. SIMONIS. *Time-Series Constraints: Improvements and Application in CP and MIP Contexts*, in "Integration of AI and OR Techniques in Constraint Programming - 13th International Conference, CPAIOR 2016, Banff, AB, Canada, May 29 - June 1, 2016, Proceedings", C. QUIMPER (editor), Lecture Notes in Computer Science, Springer, 2016, vol. 9676, pp. 18–34, https://doi.org/10.1007/978-3-319-33954-2_2

[37] A. F. BARCO, J. FAGES, É. VAREILLES, M. ALDANONDO, P. GABORIT. *Open Packing for Facade-Layout Synthesis Under a General Purpose Solver*, in "Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings", G. PESANT (editor), Lecture Notes in Computer Science, Springer, 2015, vol. 9255, pp. 508–523, http://dx.doi.org/10.1007/978-3-319-23219-5_36

[38] N. BELDICEANU, M. CARLSSON, R. DOUENCE, H. SIMONIS. *Using finite transducers for describing and synthesising structural time-series constraints*, in "Constraints", 2016, vol. 21, n^{o} 1, pp. 22–40, http://dx.doi.org/10.1007/s10601-015-9200-3

[39] S. BERARDI, M. BEZEM, T. COQUAND. *On the computational content of the axiom of choice*, in "The Journal of Symbolic Logic", 1998, vol. 63, n^{o} 02, pp. 600–622

[40] M. BEZEM, T. COQUAND, S. HUBER. *A model of type theory in cubical sets*, in "Preprint, September", 2013

[41] S. BOULIER, P.-M. PÉDROT, N. TABAREAU. *The next 700 syntactical models of type theory*, in "Certified Programs and Proofs (CPP 2017)", Paris, France, January 2017, pp. 182 - 194 [*DOI :* 10.1145/3018610.3018620], https://hal.inria.fr/hal-01445835

[42] E. BRADY. *Idris, a general-purpose dependently typed programming language: Design and implementation*, in "J. Funct. Program.", 2013, vol. 23, n^{o} 5, pp. 552–593, https://doi.org/10.1017/S095679681300018X

[43] F. CHYZAK, A. MAHBOUBI, T. SIBUT-PINOTE, E. TASSI. *A Computer-Algebra-Based Formal Proof of the Irrationality of ζ(3)*, in "Interactive Theorem Proving", R. G. GERWIN KLEIN (editor), Lecture Notes in Computer Science, Springer, 2014, vol. 8558

[44] J. COHEN, A. AJOULI. *Practical Use of Static Composition of Refactoring Operations*, in "Proceedings of the 28th Annual ACM Symposium on Applied Computing", SAC '13, ACM, 2013, pp. 1700–1705, http://dx.doi.org/10.1145/2480362.2480684

[45] J. COHEN. *Renaming Global Variables in C Mechanically Proved Correct*, in "Proceedings of the Fourth International Workshop on Verification and Program Transformation, Eindhoven, The Netherlands, 2nd April 2016", G. HAMILTON, A. LISITSA, A. P. NEMYTYKH (editors), Electronic Proceedings in Theoretical Computer Science, Open Publishing Association, 2016, vol. 216, pp. 50-64, http://dx.doi.org/10.4204/EPTCS.216.3

[46] C. COHEN, T. COQUAND, S. HUBER, A. MÖRTBERG. *Cubical Type Theory: a constructive interpretation of the univalence axiom*, 2016, To appear in post-proceedings of Types for Proofs and Programs (TYPES 2015)

[47] P. COHEN, M. DAVIS. *Set theory and the continuum hypothesis*, WA Benjamin New York, 1966

[48] J. COHEN, R. DOUENCE, A. AJOULI. *Invertible Program Restructurings for Continuing Modular Maintenance*, in "Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on", March 2012, pp. 347-352, http://dx.doi.org/10.1109/CSMR.2012.42

[49] W. R. COOK. *On understanding data abstraction, revisited*, in "Proceedings of the 24th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2009, October 25-29, 2009, Orlando, Florida, USA", S. ARORA, G. T. LEAVENS (editors), ACM, 2009, pp. 557–572, http://doi.acm.org/10.1145/1640089.1640133

[50] COQ DEVELOPMENT TEAM, THE. *The Coq proof assistant reference manual*, 2015, Version 8.5, http://coq.inria.fr

[51] P.-L. CURIEN, M. FIORE, G. MUNCH-MACCAGNONI. *A Theory of Effects and Resources: Adjunction Models and Polarised Calculi*, in "Proc. POPL", 2016, http://dx.doi.org/10.1145/2837614.2837652

[52] P.-L. CURIEN, H. HERBELIN. *The duality of computation*, in "ACM SIGPLAN Notices", 2000, vol. 35, pp. 233–243

[53] D. DELAHAYE, M. MAYERO. *Dealing with algebraic expressions over a field in Coq using Maple*, in "J. Symbolic Comput.", 2005, vol. 39, n^{o} 5, pp. 569–592, Special issue on the integration of automated reasoning and computer algebra systems, http://dx.doi.org/10.1016/j.jsc.2004.12.004

[54] R. DOUENCE, X. LORCA, N. LORIANT. *Lazy Composition of Representations in Java*, in "Software Composition, 8th International Conference, SC 2009, Zurich, Switzerland, July 2-3, 2009. Proceedings", A. BERGEL, J. FABRY (editors), Lecture Notes in Computer Science, Springer, 2009, vol. 5634, pp. 55–71, https://doi.org/10.1007/978-3-642-02655-3_6

[55] T. EHRHARD. *Call-by-push-value from a linear logic point of view*, in "European Symposium on Programming Languages and Systems", Springer, 2016, pp. 202–228

[56] A. FRISCH, G. CASTAGNA, V. BENZAKEN. *Semantic Subtyping: Dealing Set-theoretically with Function, Union, Intersection, and Negation Types*, in "J. ACM", September 2008, vol. 55, n^{o} 4, pp. 19:1–19:64

[57] C. FÜHRMANN. *Direct Models for the Computational Lambda Calculus*, in "Electr. Notes Theor. Comput. Sci.", 1999, vol. 20, pp. 245-292

[58] J.-Y. GIRARD. *Linear Logic*, in "Theoretical Computer Science", 1987, vol. 50, pp. 1-102

[59] J.-Y. GIRARD, A. SCEDROV, P. J. SCOTT. *Normal Forms and Cut-Free Proofs as Natural Transformations*, in "in : Logic From Computer Science, Mathematical Science Research Institute Publications 21", Springer-Verlag, 1992, pp. 217–241

[60] G. GONTHIER, A. ASPERTI, J. AVIGAD, Y. BERTOT, C. COHEN, F. GARILLOT, S. ROUX, A. MAHBOUBI, R. O'CONNOR, S. OULD BIHA, I. PASCA, L. RIDEAU, A. SOLOVYEV, E. TASSI, L. THÉRY. *A Machine-Checked Proof of the Odd Order Theorem*, in "Interactive Theorem Proving", S. BLAZY, C. PAULIN-

MOHRING, D. PICHARDIE (editors), Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, vol. 7998, pp. 163-179, http://dx.doi.org/10.1007/978-3-642-39634-2_14

[61] G. GONTHIER. *Formal proofs—the four-colour theorem*, in "Notices of the AMS", 2008, vol. 55, n^{o} 11, pp. 1382-1393

[62] G. GONTHIER, A. MAHBOUBI, E. TASSI. *A Small Scale Reflection Extension for the Coq system*, Inria, 2008, n^{o} RR-6455, The Reference Manual of the Ssreflect extension to the Coq tactic language, available at http://hal.inria.fr/inria-00258384

[63] T. G. GRIFFIN. *A Formulae-as-Types Notion of Control*, in "Seventeenth Annual ACM Symposium on Principles of Programming Languages", ACM Press, 1990, pp. 47–58

[64] Y. GUÉHÉNEUC, R. DOUENCE, N. JUSSIEN. *No Java without Caffeine: A Tool for Dynamic Analysis of Java Programs*, in "17th IEEE International Conference on Automated Software Engineering (ASE 2002), 23-27 September 2002, Edinburgh, Scotland, UK", IEEE Computer Society, 2002, 117 p. , https://doi.org/10.1109/ASE.2002.1115000

[65] T. C. HALES, M. ADAMS, G. BAUER, D. T. DANG, J. HARRISON, T. L. HOANG, C. KALISZYK, V. MAGRON, S. MCLAUGHLIN, T. T. NGUYEN, T. Q. NGUYEN, T. NIPKOW, S. OBUA, J. PLESO, J. RUTE, A. SOLOVYEV, A. H. T. TA, T. N. TRAN, D. T. TRIEU, J. URBAN, K. K. VU, R. ZUMKELLER. *A formal proof of the Kepler conjecture*, in "CoRR", 2015, vol. abs/1501.02155, http://arxiv.org/abs/1501.02155

[66] H. A. HELFGOTT. *The ternary Goldbach conjecture is true*, in "ArXiv e-prints", December 2013

[67] H. HERBELIN. *A Constructive Proof of Dependent Choice, Compatible with Classical Logic*, in "LICS 2012 - 27th Annual ACM/IEEE Symposium on Logic in Computer Science", Dubrovnik, Croatia, IEEE Computer Society, June 2012, pp. 365-374, https://hal.inria.fr/hal-00697240

[68] H. HERBELIN, É. MIQUEY. *Toward dependent choice: a classical sequent calculus with dependent types*, in "TYPES 2015", 2015

[69] T. HIRSCHOWITZ. *Cartesian closed 2-categories and permutation equivalence in higher-order rewriting*, in "Logical Methods in Computer Science", 2013, vol. 9, n^{o} 3, 10 p. , 19 pages [*DOI :* 10.2168/LMCS-9(3:10)2013], https://hal.archives-ouvertes.fr/hal-00540205

[70] F. IMMLER. *Verified Reachability Analysis of Continuous Systems*, in "Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings", C. BAIER, C. TINELLI (editors), Lecture Notes in Computer Science, Springer, 2015, vol. 9035, pp. 37–51

[71] G. JABER, G. LEWERTOWSKI, P.-M. PÉDROT, M. SOZEAU, N. TABAREAU. *The Definitional Side of the Forcing*, in "Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016", 2016, pp. 367–376

[72] G. JABER, N. TABAREAU, M. SOZEAU. *Extending type theory with forcing*, in "Logic in Computer Science (LICS), 2012", IEEE, 2012, pp. 395–404

[73] C. B. JAY, N. GHANI. *The Virtues of Eta-Expansion*, in "J. Funct. Program.", 1995, vol. 5, n^{o} 2, pp. 135-154

[74] U. KOHLENBACH. *Applied proof theory: proof interpretations and their use in mathematics*, Springer Science & Business Media, 2008

[75] J.-L. KRIVINE. *Realizability algebras II : new models of ZF + DC*, in "Logical Methods in Computer Science", 2012, vol. 8, n^{o} 1

[76] J. LAMBEK, P. J. SCOTT. *Introduction to higher order categorical logic*, Cambridge University Press, New York, NY, USA, 1986

[77] S. M. LANE, I. MOERDIJK. *Sheaves in Geometry and Logic*, Springer-Verlag, 1992

[78] R. LEPIGRE. *A classical realizability model for a semantical value restriction*, in "European Symposium on Programming Languages and Systems", Springer, 2016, pp. 476–502

[79] X. LEROY. *Formal certification of a compiler back-end or: programming a compiler with a proof assistant*, in "ACM SIGPLAN Notices", 2006, vol. 41, n^{o} 1, pp. 42–54

[80] P. B. LEVY. *Call-By-Push-Value: A Functional/Imperative Synthesis*, Semantic Structures in Computation, Springer, 2004, vol. 2

[81] P. B. LEVY. *Contextual isomorphisms*, in "Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages", ACM, 2017, pp. 400–414

[82] C. LIANG, D. MILLER. *Focusing and polarization in linear, intuitionistic, and classical logics*, in "Theor. Comput. Sci.", 2009, vol. 410, n^{o} 46, pp. 4747-4768

[83] Z. LUO, S. SOLOVIEV, T. XUE. *Coercive subtyping: Theory and implementation*, in "Inf. Comput.", 2013, vol. 223, pp. 18–42

[84] J. LURIE. *Higher topos theory*, Annals of mathematics studies, Princeton University Press, Princeton, N.J., Oxford, 2009

[85] S. MAC LANE. *Natural associativity and commutativity*, in "Selected Papers", 1979, pp. 415–433

[86] P. MARTIN-LÖF. *An intuitionistic theory of types: predicative part*, in "Logic Colloquium '73", 1975, vol. Studies in Logic and the Foundations of Mathematics, n^{o} 80, pp. 73–118

[87] P.-A. MELLIÈS. *Asynchronous Games 3 An Innocent Model of Linear Logic*, in "Electr. Notes Theor. Comput. Sci.", 2005, vol. 122, pp. 171-192

[88] P.-A. MELLIÈS. *Asynchronous Games 4: A Fully Complete Model of Propositional Linear Logic*, in "LICS", 2005, pp. 386-395

[89] P.-A. MELLIÈS, N. TABAREAU. *Resource modalities in tensor logic*, in "Ann. Pure Appl. Logic", 2010, vol. 161, n^{o} 5, pp. 632-653

[90] É. MIQUEY. *A classical sequent calculus with dependent types*, in "European Symposium on Programming", Springer, 2017, pp. 777–803

[91] E. MOGGI. *Computational lambda-calculus and monads*, in "Proceedings of the Fourth Annual IEEE Symposium on Logic in Computer Science (LICS 1989)", IEEE Computer Society Press, June 1989, pp. 14–23

[92] E. MOGGI. *Notions of computation and monads*, in "Inf. Comput.", July 1991, vol. 93, n^{o} 1, pp. 55–92, http://dx.doi.org/10.1016/0890-5401(91)90052-4

[93] G. MUNCH-MACCAGNONI. *Formulae-as-Types for an Involutive Negation*, in "Proceedings of the joint meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (CSL-LICS)", 2014

[94] G. MUNCH-MACCAGNONI. *Models of a Non-Associative Composition*, in "Proc. FoSSaCS", A. MUSCHOLL (editor), LNCS, Springer, 2014, vol. 8412, pp. 397–412

[95] G. MUNCH-MACCAGNONI. *Note on Curry's style for Linear Call-by-Push-Value*, 2017, https://hal.inria.fr/hal-01528857

[96] A. NANEVSKI, G. MORRISETT, A. SHINNAR, P. GOVEREAU, L. BIRKEDAL. *Ynot: Reasoning with the awkward squad*, 2008

[97] C. OKASAKI. *Purely functional data structures*, Cambridge University Press, 1999

[98] C. PRUD'HOMME, X. LORCA, R. DOUENCE, N. JUSSIEN. *Propagation engine prototyping with a domain specific language*, in "Constraints", 2014, vol. 19, n^{o} 1, pp. 57–76, https://doi.org/10.1007/s10601-013-9151-5

[99] C. PRUD'HOMME. *Contrôle de la propagation et de la recherche dans un solveur de contraintes. (Controlling propagation and search within a constraint solver)*, École des mines de Nantes, France, 2014, https://tel.archives-ouvertes.fr/tel-01060921

[100] P.-M. PÉDROT, N. TABAREAU. *An Effectful Way to Eliminate Addiction to Dependence*, January 2017, https://hal.inria.fr/hal-01441829

[101] K. QUIRIN, N. TABAREAU. *Lawvere-Tierney sheafification in Homotopy Type Theory*, in "Journal of Formalized Reasoning", 2016, vol. 9, n^{o} 2 [*DOI :* 10.6092/ISSN.1972-5787/6232], https://hal.inria.fr/hal-01451710

[102] J. C. REYNOLDS. *Types, Abstraction and Parametric Polymorphism*, in "IFIP Congress", 1983, pp. 513-523

[103] P. SELINGER. *Control Categories and Duality: On the Categorical Semantics of the Lambda-Mu Calculus*, in "Math. Struct in Comp. Sci.", 2001, vol. 11, n^{o} 2, pp. 207–260

[104] S. G. SIMPSON. *Subsystems of Second Order Arithmetic*, Second, Cambridge University Press, 2009, Cambridge Books Online, http://dx.doi.org/10.1017/CBO9780511581007

[105] K. STØVRING. *Extending the Extensional Lambda Calculus with Surjective Pairing is Conservative*, in "Logical Methods in Computer Science", 2006, vol. 2, n^{o} 2

[106] N. SWAMY, C. HRIŢCU, C. KELLER, A. RASTOGI, A. DELIGNAT-LAVAUD, S. FOREST, K. BHARGAVAN, C. FOURNET, P.-Y. STRUB, M. KOHLWEISS, J.-K. ZINZINDOHOUE, S. ZANELLA-BÉGUELIN. *Dependent Types and Multi-Monadic Effects in F**, in "43nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)", ACM, January 2016, pp. 256-270, https://www.fstar-lang.org/papers/mumon/

[107] É. TANTER, N. TABAREAU. *Gradual Certified Programming in Coq*, in "Proceedings of the 11th ACM Dynamic Languages Symposium (DLS 2015)", Pittsburgh, PA, USA, ACM Press, October 2015, pp. 26–40

[108] UNIVALENT FOUNDATIONS PROJECT. *Homotopy Type Theory: Univalent Foundations for Mathematics*, http://homotopytypetheory.org/book, 2013

[109] M. VÁKÁR. *A Framework for Dependent Types and Effects*, in "arXiv preprint arXiv:1512.08009", 2015

[110] B. ZILIANI, D. DREYER, N. R. KRISHNASWAMI, A. NANEVSKI, V. VAFEIADIS. *Mtac: A monad for typed tactic programming in Coq*, in "Journal of Functional Programming", 2015, vol. 25, http://dx.doi.org/10.1017/S0956796815000118

[111] F. VAN RAAMSDONK. *Higher-order Rewriting*, in "Proc. Rewrit. Tech. App.", LNCS, Springer, 1999, vol. 1631, pp. 220-239