# 2023

# DOSSIER DE CANDIDATURE

# *APPLICATION*

**Cochez le concours sur lequel vous candidatez**
**Check the competition exam for which you are applying**

☒ **ISFP - (Inria Starting Faculty Position / Inria Starting Faculty Position)**

☒ **CRCN - (Chargés de recherche de classe normale / Young graduate scientist position)**

☐ **DR2 - (Directeurs de recherche de deuxième classe / Senior researcher position)**

**Nom**[1] **:** FORSTER
***Last name***

**Prénom :** Yannick
***First name***

**Sexe :** ☐ **F** ☒ **M**
***Sex***

**Nom utilisé pour vos publications (facultatif) :**
***Name used for your publications (optional):***

[1] Il s'agit du nom usuel figurant sur vos pièces d'identité
It is the name appearing on your identity cards

## DEPOT DE VOTRE CANDIDATURE
## SUBMITTING YOUR APPLICATION

**Le dossier de candidature doit comprendre :**

- Formulaire 1 : Parcours professionnel
- Formulaire 2 : Description synthétique de l'activité antérieure
- Formulaire 3 : Contributions majeures
- Formulaire 4 : Programme de recherche
- Formulaire 5 : Liste complète des contributions

CRCN & ISFP :

- Les rapports de thèse ou de doctorat (si disponibles)
- Une copie des derniers titres et diplômes
- Une photographie récente de la candidate / du candidat (facultative)

**The application file must include:**

- *Form 1: Professional history*
- *Form 2: Summary of your past activity*
- *Form 3: Major contributions*
- *Form 4: Research program*
- *Form 5: Complete list of contributions*

*CRCN & ISFP:*

- *PhD dissertation reports –* ***not available in Germany***
- *A copy of most recent titles and diplomas –* ***attached***
- *A recent photography of the applicant (optional)*

**SOMMAIRE** / ***SUMMARY***

**Formulaire 1 — PARCOURS PROFESSIONNEL**
***Form 1 — PROFESSIONAL HISTORY***

## 1) Parcours Professionnel / *Professional history*

**Situation professionnelle actuelle** / ***Current professional status***

Statut et fonction[2] / *Position and Status[2]*: Postdoc on Marie Skłodowska-Curie fellowship
Etablissement (ville - pays) / *Institution (city - country)*: Inria Rennes Bretagne Atlantique, Gallinette Project-Team, Nantes, France
Date d'entrée en fonction / *Start*: 01.12.2021
[ ] Sans emploi / *Without employment*

**Expériences professionnelles antérieures** /***Previous professional experiences***

Déroulez votre parcours professionnel antérieur à Inria, chez Inria, en détachement ou en mise à disposition.
*Detail your professional history, before Inria, at Inria, on secondment or leave.*

| Date début<br>*Start* | Date fin<br>*End* | Etablissement<br>*Institution* | Fonction et statut[2]<br>*Position and status[2]* |
|---|---|---|---|
| 30.09.2016 | 30.11.2021 | Saarland University | research assistant / PhD student |

Nombre d'années d'exercice des métiers de la recherche après la thèse / *Number of years of professional research experience after the PhD:* 1 year, 3 months (as of March 5th, 2023)

## 2) Interruptions de carrière/*Career breaks*

None

## 3) Encadrement d'étudiants et de jeunes chercheurs / *Supervision of students and early-stage researchers*

Encadrement de thèses, postdocs, stages (master ou autres). Indiquez le nom des personnes encadrées, le sujet de leurs travaux, la part prise dans leur encadrement, et présentez brièvement le contenu et la portée de ces travaux. Fournissez une URL vers les thèses concernées.
*Supervision of PhDs, postdocs, and interns (master's or others). Mention the names of students, their research subjects, the amount of supervision involved, and give a brief presentation of the contents and significance of the work. Provide a URL to the relevant PhD theses.*

**Supervised Bachelor's theses at Saarland University (1 semester preparation + 3 months thesis phase)**

2022 Niklas Mück: "The Arithmetical Hierarchy, Oracle Computability, and Post's Theorem in Synthetic Computability", co-supervised with Dominik Kirst. Lead to TYPES 2022 abstract [22].

2020 Felix Jahn: "Synthetic one-one, many-one, and truth-table reductions in Coq". Lead to TYPES '22 abstract [21], CPP '23 paper [26], and CSL '23 paper [25].

2019 Marcel Ullrich: "Generating induction principles in MetaCoq". Lead to CoqWS '20 presentation [W8].

2018 Dominik Wehr: "A Constructive Analysis of First-Order Completeness Theorems in Coq", co-supervised with Dominik Kirst. Lead to LFCS '20 paper [10].

2018 Simon Spies: "Undecidability of Higher-Order Unification in Coq". Lead to CPP '20 paper [13].

2017 Maximilian Wuttke: "Verified Programming of Turing Machines in Coq". Lead to CPP '20 paper [14].

2017 Edith Heiter: "Undecidability of PCP in Coq", co-supervised w/ Gert Smolka. Lead to ITP '18 paper [3].

**Supervised Master's theses at Saarland University (1 semester preparation + 6 months thesis phase)**

2022 Roberto Álvarez Castro: "Mechanized undecidability of subtyping in System F".

**Supervised Internships at Saarland University**

2020 Bohdan Liesnikov: “Generating subterm relations in MetaCoq”. 3 months internship, resulted in Coq workshop abstract [58].

2018 Simon Spies: “Denotational Semantics of CBPV in Coq”. 2 months internship, resulted in CPP ’19 paper [47]

**Supervised Internships at Gallinette Team, Inria, Nantes**

2022 Nils Lauermann: “Verified translation of coinductive to inductive types in MetaCoq”. 6 months internship, ongoing.

2021 Tomas Vallejos Parada: “Generating proofs of the fundamental lemma of the parametricity translation in MetaCoq”. 6 months internship.

## 4) Encadrement de développements technologiques (logiciel, matériel, robotique) / *Supervision of technological development (software, hardware, robotics)*

None

## 5) Responsabilités collectives / *Responsibilities*

2023 Workshop co-chair: 28th ACM SIGPLAN International Conference on Functional Programming (ICFP 2023)
https://icfp23.sigplan.org/committee/icfp-2023-organizing-committee

2023 Program Committee: 13th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2023)
https://popl23.sigplan.org/committee/CPP-2023-papers-program-committee

2022 Member of the operations team of the ACM SIGPLAN Long-Term Mentoring Committee (SIGPLAN-M), ongoing
https://www.sigplan.org/LongTermMentoring/

2021 Program Committee: 16th Logical and Semantic Frameworks with Applications (LSFA 2021)
https://mat.unb.br/lsfa2021/pages/committees.html

2021 Program Committee: 11th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2021)
https://popl21.sigplan.org/committee/CPP-2021-program-committee

**External Reviewer**

- 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2023)
- 31st European Symposium on Programming (ESOP 2022)
- 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2021)
- Fundamentae Informaticae (2021)
- 6th International Conference on Formal Structures for Computation and Deduction (FSCD 2021)
- 10th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020)
- 9th Coq Workshop (2019)
- 10th Internation Conference on Interactive Theorem Proving (ITP 2019)
- 21st International Symposium on Principles and Practice of Declarative Programming (PPDP 2019)
- 4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)
- 25th Workshop on Logic, Language, Information and Computation (WoLLIC 2018)
- 10th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2018)

## 6) Management (si pertinent) / *Management*

Coordinating the work of Kazuhiko Sakaguchi (research engineer Gallinette team) and Nils Lauermann (6 months intern) on verifying Coq’s extraction from MetaCoq.

## 7) Mobilité (si pertinent) / *Mobility (if relevant)*

2016–2021 PhD at Saarland University, Germany. Thesis supervised by Prof. Dr. Gert Smolka.

2017 One month long research visit at the University of Oxford, UK, working with Ohad Kammar.

2015–2016 Master in Advanced Computer Science at Cambridge University, UK. Thesis supervised by Marcelo Fiore.

## 8) Enseignement (si pertinent) / *Teaching (if relevant)*

**For the Parisian Master of Research in Computer Science (MPRI)**

Winter 2022 Assistants de preuves, Lecturer, with Matthieu Sozeau and Théo Winterhalter, in total 8 lectures of 3 hours length

**At Saarland University**

Winter 2020 Advanced Coq Programming, Lead Organiser and Lecturer, Programming Systems Lab, 2 week full day block course with daily lectures and tutorials plus 6 week project phase

Winter 2018 Programming 1, Course management, Programming Systems Lab, 16 week course with $\sim$ 550 students and a team of 17 tutors. Weekly exercise sheets, weekly tests, two exams.

Summer 2017 Mathematics Preparatory Course for CS students, Lead Organiser, CS department, 4 week course for 250 students with a team of around 40 volunteers

Summer 2018 Advanced Coq Programming, Organiser and Lecturer, Programming Systems Lab, 2 week full day block course with daily lectures and tutorials, plus 6 week project phase

Summer 2017 Didactic Seminar for Student TAs in Programming 1, Organiser, Reactive Systems Group, 3 days full day seminar for a team of 15 tutors

Summer 2016 Mathematics Preparatory Course for CS students, Lead Organiser and Lecturer, CS department, 4 week course for 250 students with a team of around 40 volunteers

Summer 2015 Mathematics Preparatory Course for CS students, Organisation team, CS department, 4 week course for 250 students with a team of around 40 volunteers

Winter 2014 Didactic Seminar for Re-exam TAs in Programming 1, Organiser, Reactive Systems Group, 3 days full day seminar for a team of 4 tutors

## 9) Diffusion de l'information scientifique (si pertinent) / *Dissemination of scientific knowledge (if relevant)*

2018 Research Days Computer Science at Saarland University: 3 full-day courses on Coq for high school student winners of the German Federal Computer Science Competition

## 10) Visibilité (si pertinent) / *Visibility (if relevant)*

2023 Invited talk in special session on use of proof assistants at 39th Conference on Mathematical Foundations of Programming Semantics MFPS XXXIX (MFPS 2023)

2023 Invited lecture series at the Proof and Computation Autumn School, Herrsching, September 11th-17th. Organizers: Klaus Mainzer, Peter Schuster, Helmut Schwichtenberg. https://www.mathematik.uni-muenchen.de/~schwicht/pc23.php

2022 "MetaCoq as a tool to prevent future unsoundness in Coq". Invited talk at the workshop on Sources of Unsoundness in Verification (Unsound 2022), co-located with SPLASH '22, December 6th, Auckland, New Zealand (online). Joint work with the MetaCoq team.

2022 "Synthetic Computability in Constructive Type Theory". Talk at the Chocola meeting. June 2nd 2022, Lyon. Joint work with Dominik Kirst, Gert Smolka, Felix Jahn, Niklas Mück, Nils Lauermann, Fabian Kunze, and the contributors of the Coq Library of Undecidability Proofs.

2022 "Verified Extraction to OCaml from Coq, in Coq." Invited Talk at the Conference on Algorithmic Law Design and Implementation. April 28th 2022, Barcelona, Spain. Joint work with Matthieu Sozeau, Pierre Giraud, Pierre-Marie Pédrot, and Nicolas Tabareau.

2022 Rosser prize (best student paper award) at the Symposium on Logical Foundations of Computer Science for "Parametric Church's Thesis: Synthetic Computability Without Choice", https://lfcs.ws.gc.cuny.edu/lfcs-2022

## Formulaire 2 — DESCRIPTION SYNTHÉTIQUE DE L'ACTIVITÉ ANTÉRIEURE

*Form 2 — SUMMARY OF YOUR PAST ACTIVITY*

My lines of research are in the intersection of type theory, interactive theorem proving, computability theory, constructive (reverse) mathematics, and logical foundations. I focus on increasing the trustworthiness of both interactive theorem provers (ITPs) such as Coq and programs written in ITPs and on improving the maintainability of both ITPs and proofs therein.
ITPs have been successfully used to obtain high trust in critical programs such as e.g. the CompCert compiler for C or elliptic curves in Google's BoringSSL library, and in landmark results of mathematics (e.g. the Four colour theorem, the Feit-Thompson theorem, the Kepler conjecture, or the Scholze-Clausen theorem). While the use of ITPs seems to be on the rise, especially among mathematicians and programming language researchers, the question how ITPs can be used sustainably in daily practice with low long-term maintenance overhead are open research problems.
I decidedly maintain a separate line of research connected to formalise mathematics in ITPs. This allows me to understand the user perspective on ITPs and allows broader international collaborations.

**Verifying extraction** I am part of the MetaCoq team (a project specifying and formalising the type theory of Coq in Coq) [66, 67], the CertiCoq team (a project verifying a compiler from Coq to C in Coq), and I work on replacing the extraction process of Coq (creating OCaml programs from Coq programs) by a new implementation verified in Coq as part of my current Marie Skłodowska-Curie Postdoctoral Fellowship [49]. For MetaCoq, I have proved the first phase of extraction, namely type and proof erasure correct [67]. For CertiCoq, I have contributed several passes related to eta-expanding constructors that allowed closing the last proofs in the CertiCoq project. For verified extraction to OCaml, I have specified the target language in Coq and am close to finishing the operational correctness proof of the extraction process.

**Modular proofs in Coq** I work on techniques to develop proofs in a modular manner, i.e. on the expression problems for verified programs and proofs [50], with Kathrin Stark (Heriot Watt University) and Kenji Maillard (Inria Nantes). While meta-programming for programming languages is a well studied topic, meta-programming languages for proofs are still underdeveloped despite having a lot of unused potential. I thus work on making MetaCoq's meta-programming facilities, which can be used from Coq directly without having to learn or consider a new programming language or paradigm, more powerful and accessible. I put emphasis on identifying student projects and including students in these projects, yielding insights w.r.t. accessibility while simultaneously enabling them to experience research early.

**Synthetic computability and constructive reverse mathematics** Additionally, I decidedly follow a third, more foundational line of research in constructive mathematics, computability, and type theory [33, 29, 31, 30]. Concretely, I work on constructive reverse mathematics of statements in computability theory through the lens of synthetic computability theory. In synthetic computability theory, proofs can be developed in full formality without having to formally deal with models of computation. This line has a lower entrance barrier for students, and I am working towards using it in teaching where ITPs can be used to ease the learning curve in theoretical lectures for more practically minded students [22]. As a result of this work, I co-founded and co-maintain the Coq Library of Undecidability proofs with Dominique Larchey-Wendling (LORIA Nancy) and Andrej Dudenhefner (Uni Dortmund) [46], one of the largest libraries of formal mathematical proofs in the Coq ecosystem, with the landmark result of the first machine-checked undecidability proof of Diophantine equations, i.e. Hilbert's tenth problem [65].

**Dormant line: Complexity theory in Coq** Complexity theory is maybe the topic of (theoretical) computer science which received the least attention in publications of the interactive theorem proving community. This can be explain due to the de Bruin factor of results in complexity theory being extremely high, i.e. the ratio of the length of a proof on paper and of a machine-checked proof in an ITP. In complexity theory, proofs rarely go on the level of the model of computation, and never spell out exact runtime functions, but rather work with intuition based on Landau notation. With Fabian Kunze and Marc Roth I have proposed the weak call-by-value $\lambda$-calculus as reasonable model of computation subject to computer-formalised complexity theory: We have shown that this calculus and Turing machines can simulate each other with a polynomial overhead in time and a constant factor overhead in space, a result that was long open and not anticipated. I have been subsequently involved in the founding of the Coq library of complexity theory, where I co-contributed a machine-checked equivalence proof of Turing machines and the weak call-by-value $\lambda$-calculus w.r.t. time complexity.

**Dormant line: CBPV** My Master's thesis was advised by Ohad Kammar and supervised by Marcelo Fiore and concerned with showing that algebraic effects and monadic reflection based on the call-by-push value calculus can simulate each other, but are strictly differently expressive when enforcing a simple type discipline. We developed this into an ICFP paper with Sam Lindley and Matija Pretnar [35] and a subsequent extended version in the journal of functional programming [36].

## Formulaire 3 — CONTRIBUTIONS MAJEURES
*Form 3 — MAJOR CONTRIBUTIONS*

**Taille maximum de cette partie / Maximum size for this part :**

- **CRCN & ISFP – 3 fiches**
  **Taille maximum de cette partie : 3 pages**
  ***Maximum size for this part: 3 pages***

Remplir une fiche par contribution majeure (5 au plus pour les candidatures DR2 — 3 au plus pour les candidatures CRCN & ISFP).

Il peut s'agir d'une contribution scientifique donnant lieu à un ensemble de publications (voire une publication majeure), ou à un développement technologique (logiciel, matériel, robotique ou autre), d'une action de transfert industriel ou sociétal, d'une responsabilité collective, d'une activité d'animation d'une communauté de recherche, ou tout autre élément relevant des missions d'un chercheur ou d'une chercheuse. Les critères importants sont la créativité, l'originalité et l'impact. Chaque fiche suivra le plan indiqué ci-dessous. Dans l'ensemble du texte, pensez à donner, le cas échéant, les références permettant de consulter sur le Web les documents mentionnés (articles, thèses, logiciels, etc.).

Pour les logiciels, fournissez une autoappréciation selon le canevas disponible dans le document « Criteria for Software Self-Assessment » disponible à l'URL
https://www.inria.fr/sites/default/files/2021-01/Criteria%20software%20self%20assessment.pdf.

Pour les actions de transfert (transfert technologique ou sociétal), fournissez une description selon le canevas disponible dans le document « Evaluation des contributions scientifiques en matière de transfert / Guide méthodologique » disponible à l'URL
https://www.inria.fr/sites/default/files/2020-01/2018-06-GuideMethodologique_EvaluationTransfert%281%29.pdf.

Fill in one form for each major contribution (at most 5 for the DR2 candidates — 3 for the CRCN & ISFP candidates).

It may be a scientific contribution expressed through a set of publications (or a single major publication) or through a technological development (software, hardware, robotic, or other); it may also be an industrial or a societal transfer, a participation to the management of research or to the animation of a scientific community, or any other element. The main criteria are creativity, originality and impact. Each form should follow the guidelines given below. In the body of the text, give the Web references for quoted documents (articles, dissertations, software,...), if available.

For software, please use the « Self-assessment software criteria » guideline, available at the URL
https://www.inria.fr/sites/default/files/2021-01/Criteria%20software%20self%20assessment.pdf.

For transfer actions (technology or society transfer) please describe your achievements following the guidelines « Evaluating scientific contributions in relation to transfer / Methodological guide » available at the URL
https://www.inria.fr/sites/default/files/2020-01/2018-06-GuideMethodologique-EvaluationTransfert_EN%281%29.pdf.

## Fiche 1 : Verified Operational Correctness of Extraction from Coq to OCaml

### 1. Description de la contribution / *Description of the contribution*

One of the central claims of fame of the Coq proof assistant is extraction, i.e. the ability to obtain efficient programs in industrial programming languages such as OCaml from programs written in Coq's expressive dependent type theory, used, amongst many others, in the CompCert project. For executables obtained by extraction, the extraction process is part of the TCB, as are Coq's elaboration and kernel, and the compiler used to compile the extracted code. The MetaCoq project aims at decreasing the TCB of Coq's kernel by re-implementing it in Coq itself and proving it correct w.r.t. a formalisation of Coq's type theory in Coq. As part of this project, we have contributed a verified type and proof erasure procedure [63] to the intermediate calculus $\lambda_{\square}$, mirroring Coq's syntax closely but having a constructor $\square$ indicating that a type or proof was erased. We aim at taking one step further, namely removing the compilation from $\lambda_{\square}$ to OCaml from the TCB as well. Our goal is to provide a verified drop-in replacement for Coq's extraction process to OCaml. To be able to start with a trustworthy specification, we extract to the intermediate language of the OCaml compiler as specified in the Malfunction project [24].

### 2. Contribution personnelle de la candidate ou du candidate / *Personal contribution of the applicant*

I have given the first verified correctness proof of type and proof erasure from Coq to $\lambda_{\square}$, in Coq. Subsequently, the proof has been altered and generalised by collaborators. Furthermore, I have implemented or co-implemented and verified or co-verified all 10 intermediate passes between Coq and OCaml.

### 3. Originalité et difficulté / *Originality and difficulty*

There are several ongoing projects on obtaining correct programs in industrial programming languages like OCaml, Haskell, or C from programs in richly typed dependent programming languages like Coq, Idris, and Agda [21], or the simply type theories underlying Isabelle/HOL or HOL4 [53, 60].
This project is closely related to all of them, but has a separable motivation: We want to replace the current extraction process of Coq to OCaml [57] with a verified process, without causing maintenance issues in the projects using extraction. Similar to the current extraction process, we require the generated programs to be readable, amongst other reasons to allow developers to set up extraction directives like `Extract Constant`. In contrast, we do *not* require the generated programs to be easily modifiable: All modifications should be done on the Coq side and obtained through extraction, otherwise all guarantees about the correctness of code are lost.
In order to give a machine-checked implementation of extraction, we require a specification of the target language in Coq. This specification will remain part of the TCB: It has to be correct for the verification to be meaningful.
Thus, it is desirable to have a *clear*, easy-to-review specification. One path would be to specify the operational semantics of OCaml in Coq, or identify a suitable subset of it. Such a semantics could be obtained by translating the semantics of OCaml given in prose, or by reverse-engineering a specification from a compiler, interpreter, or an abstract machine for OCaml. It is also desirable that the chosen semantics be *economical*, in the sense that it can be used in a verification project without generating an overhead which lets even easy tasks take months.
To have a clear and economical specification, we choose as target language for extraction Malfunction [24], a specification of `lambda`, the intermediate language of the OCaml compiler, which comes both with an interpreter and a compiler.
Same as Coq, $\lambda_{\square}$ has higher-order constructors, structural fixpoints with principal arguments that have to reduce to a constructor for the fixpoint to unfold, and no dedicated reduction strategy. On the other hand, constructors in OCaml are blocks (e.g. `cons` by itself is not well-typed, only `cons(x,l)` is), `let rec` does not indicate a special principal argument, and reduction is weak call-by-value. These subtle differences pose challenges for machine-checked reasoning, since they do not occur in proofs on paper.

### 4. Validation et impact / *Validation and impact*

We aim at having the drop-in replacement for Coq's extraction ready by the end of the year, including optimisations. This change will be important for Coq, because the current extraction process is unmaintained and has several well-known problems.
Already now, the work on $\lambda_{\square}$ as intermediate language for implementation and verification has had two significant impacts: First, it is used as the front-end of the CertiCoq project. The progress towards verified extraction to OCaml and several intermediate passes now allowed for the first time to have a fully verified pipeline for CertiCoq, 7 years after the start of the project. Secondly, it is used in the ConCert project [17], which focuses on extracting from Coq into blockchain languages.

### 5. Diffusion / *Dissemination*

The verification of type and proof erasure was published at POPL '20 [63]. A report on verified extraction to OCaml was given at the ML Family Workshop '20 [49].

## Fiche 2 : Robust and Accessible Modular Proofs using Meta-Programming

### 1. Description de la contribution / *Description of the contribution*

The *expression problem*, a term coined by Philip Wadler, is concerned with reusing and extending definitions in programming languages: "The goal is to define a datatype by cases, where one can add new cases to the datatype and new functions over the datatype, without recompiling existing code." (Genericity mailing list, 1998).
In the context of machine-checked proofs, various areas also rely on the inductive characterisation of syntax, e.g. meta-theory of programming languages or logical calculi such as first-order, higher-order, separation, or temporal logics. For pen-and-paper developments in these areas it is common to extend a language by new constructs after developing certain results, and then just sketch the new cases for function definitions and proofs.
When inductive types are used in (type-theoretic) proof assistants to define syntax, type systems, deduction, etc., type definitions, function definitions, and proofs are closed to such extensions. Available approaches for modular syntax are either inapplicable to type theory or add a layer of indirectness by requiring complicated encodings of types.
In work with Kathrin Stark, we have presented a concise, transparent, and accessible approach to modular syntax with binders by adapting Swierstra's Data Types à la Carte approach to the Coq proof assistant. Our approach relies on two phases of code generation: We extend the Autosubst 2 tool and allow users to specify modular syntax with binders in a HOAS-like input language. To state and automatically compose modular functions and lemmas, we implement commands based on MetaCoq. We support modular syntax, functions, predicates, and theorems, and provide tactics to elegantly deal with modular proofs.
The practicality of our approach was demonstrated by modular proofs of preservation, weak head normalisation, and strong normalisation for several variants of mini-ML.

### 2. Contribution personnelle de la candidate / du candidat / *Personal contribution of the candidate*

The motivation to develop modular syntax in Coq came from the joint CPP contribution on a Coq development concerning CBPV with Kathrin Stark, Steven Schäfer, and Simon Spies [47]. I co-designed the Coq á la Carte approach with Kathrin Stark and was in charge of implementing the Ltac tactics and the MetaCoq-based meta-programming tools to compose functions and proofs. Furthermore, I implemented several of the case studies and co-wrote the paper.

### 3. Originalité et difficulté / *Originality and difficulty*

The Coq à la Carte approach is the first approach to modularity in proof assistants which does not require a user to understand heavy mathematics first. It is decidedly light-weight and shallow from a mathematical standpoint, and draws its strength from automation by external tools such as Autosubst, tactics based on e.g. Ltac, and meta-programming such as MetaCoq – which in turn is easy to provide due to the conceptual simplicity.
As such, it can be seen as a further contribution towards solving a long-standing open problem: How to develop proofs modularly.
Simultaneously, it was one of the first project making serious use of MetaCoq's meta-programming capabilities.

### 4. Validation et impact / *Validation and impact*

The approach has been successfully used in a student project by Roberto Alvarez, which I co-advised with Dominik Kirst. The project gave a modular presentation of propositional and first-order logic, with a deduction system and a modular soundness proof w.r.t Tarski-style semantics.

### 5. Diffusion / *Dissemination*

The Coq à la Carte approach was published at CPP 2019 in a joint paper with Kathrin Stark [51].
I have taken further steps towards sustainable meta-programming in work with Bohdan Liesnikov and Marcel Ullrich reported on at the Coq workshop 2020 [58], and in the internship by Tomas Vallejos in the Gallinette team in Nantes, which we hope can contribute to Coq à la Carte in the future.

## Fiche 3 : A basis for machine-checked cutting-edge computability proofs

### 1. Description de la contribution / *Description of the contribution*

Many areas of mathematics have been formalised in interactive theorem provers, including fields of theoretical computer science such as logic, automata, graphs, programming languages, etc. An area that has long only seen limited machine-checked results is computability theory: This is because in proofs on paper about computability theory, the Church Turing thesis is used liberally. As a consequence, programs are rarely spelled out even in pseudo-code and almost never actually defined in the underlying model of computation such as Turing machines, $\mu$-recursive functions, or $\lambda$-calculus. When translating these proofs to a proof assistant, the definitions in the model of computation have to be filled in and verified, which creates a huge overhead in time and difficulty, often surpassing the difficulty of the mathematical essence of the proof significantly.
With a so-called synthetic approach, one can introduce (consistent) axioms in constructive foundations such as constructive type theory which allow ignoring details in the model of computation and thus to focus on the mathematical essence of proofs, even when being fully formal. Synthetic computability was introduced by Richman [61] and further pioneered by Bauer [18]. However, both lines of work have in common that they are anti-classical, i.e. the law of excluded middle is provably false – an unsatisfying situation, since the law of excluded middle is omnipresent in computability proofs.
With my collaborators, I have devised an approach to synthetic computability which is elegant and machine-checkable, but compatible with classical logic. The machine-checked proofs resemble paper proofs and can focus on the mathematical essence of results.

### 2. Contribution personnelle de la candidate / du candidat / *Personal contribution of the candidate*

I co-devised the notion of synthetic undecidability, where a problem $p$ is undecidable if the decidability of $p$ would imply the decidability of the halting problem of Turing machines, with Dominique Larchey-Wendling (LORIA Nancy), Dominik Kirst, and Gert Smolka (Saarland University) [45, 37]. I analysed the relationship of the axioms "Church's thesis" (CT) to other axioms in constructive type theory [28], which ultimately lead to the observation that a combination of CT with an assumed $s$-$m$-$n$ operator allows for the development of synthetic computability in Coq's constructive type theory, while being compatible with classical logic [31].
Together with Dominique Larchey-Wendling, I founded the Coq library of undecidability proofs, which contains by now undecidability proofs for almost all problems which are commonly used at the start of proofs by reduction.
I advised the Bachelor's theses of Edith Heiter and Simon Spies as well as the Master's thesis of Roberto Alvarez on synthetic undecidability, of Maximilian Wuttke on models of computation, and of Felix Jahn and Niklas Mück on synthetic computability.

### 3. Originalité et difficulté / *Originality and difficulty*

Formalising serious amounts of computability theory surpassing Rice's theorem in a proof assistant has been a long-standing open problem.
The approach described in this fiche solves this problem with a mathematically elegant but practically applicable solution. By working in Coq's type theory, it is fully compatible with classical logic and can thus be easily adapted by external collaborators. Furthermore, inspecting Coq's type theory from the perspective of synthetic computability reveals deep foundational aspects of constructive type theory in general that were folklore in the community, but are usually not made explicit – such as the status of the unique choice axiom.

### 4. Validation et impact / *Validation and impact*

Besides the dissemination in scientific contributions described below, the approach has been used in several scientific papers without my involvement [56, 55, 27, 26, 52, 25].
It was also used in the Bachelor's thesis of Nils Lauermann at Saarland University, advised by Fabian Kunze, without my involvement.

### 5. Diffusion / *Dissemination*

An overview of the Coq library of undecidability proofs has been given at the Coq Workshop '20 [46].
We presented steps in machine-checked computability based on the call-by-value $\lambda$-calculus at ITP '17 [48], and gave the first machine-checked synthetic reduction at ITP '18[32].
We established the notion of synthetic undecidability at CPP '19 [37], and gave more undecidability proofs w.r.t this notion at CPP '19 [45] and CPP '20 [64].
The equivalence of Turing machines and $\lambda$-calculus also including time complexity was presented at ITP '21 [43].
I first analysed axioms for synthetic computability in Coq at CSL '21 [29], established a setting for (classical) synthetic computability in Coq at LFCS '22 [31], which was use to machine-check the theory of reducibility at CSL '23 [33].

**Formulaire 4 — PROGRAMME DE RECHERCHE**
***Form 4 — RESEARCH PROGRAM***

Étant donné l'organisation d'Inria, tout chercheur, ou toute chercheuse, a vocation à être affecté(e) dans une équipe-projet. Un candidat, ou une candidate, indique donc généralement dans son dossier de candidature l'équipe-projet dans laquelle elle ou il souhaite être affecté(e). Il est dans ce cas fortement recommandé de prendre préalablement contact avec la ou le responsable de l'équipe-projet souhaitée.

Il est néanmoins aussi possible de déposer une candidature sans préciser *a priori* une équipe-projet d'accueil. Dans ce cas, si la candidate ou le candidat est déclaré(e) admissible, une ou plusieurs équipes d'accueil pourront lui être proposées entre la phase d'admissibilité et la phase d'admission. Cette proposition d'affectation se fera en prenant en compte les aspirations de la candidate ou du candidat, celles des équipes, et la politique scientifique d'Inria.

Dans le cas où la candidature a lieu dans une équipe-projet, le candidat ou la candidate est invité(e) à expliquer dans son programme de recherche son intégration dans l'équipe-projet souhaitée.

Dans le cas où l'équipe-projet n'est pas choisie au moment de la candidature, le candidat ou la candidate n'est pas tenu(e) de détailler son intégration. Elle ou il peut néanmoins, sans que cela soit une obligation, indiquer des noms de chercheurs ou de chercheuses avec qui elle ou il pourrait collaborer en cas de recrutement.

☒ Je souhaite candidater dans l'équipe-projet, ou les équipes-projets suivante(s) : Cambium

☐ Je ne souhaite pas choisir d'équipe-projet pour l'instant. En cas d'admissibilité, je serai contacté(e) par la présidente ou le président du jury pour discuter de possibles équipes d'accueil.

*Given Inria's organization, any researcher should be assigned to a project-team. A candidate therefore generally indicates in his or her application file the project-team to which he or she wishes to be assigned. In this case, it is strongly recommended to contact the leader of the desired project-team beforehand.*

*However, it is also possible to submit an application without specifying a priori a host project-team. In this case, if the candidate is declared eligible, one or more host teams will be proposed between the eligibility phase and the admission phase. This assignment proposal will be made taking into account the aspirations of the candidate, those of the teams, and Inria's scientific policy.*

*In the case of an application in a project-team, the candidate is invited to explain in his or her research program the integration in the desired project-team.*

*In the case when the project-team is not selected at the time of the application, the candidate is not required to detail his or her integration. However, he or she may, without this being an obligation, indicate the names of researchers with whom he or she could collaborate in the case of recruitment.*

☒ *I would like to apply for the following project-team(s): Cambium*

☐ *I prefer not to choose a project-team for the moment. If I am considered eligible, I will be contacted by the chair of the jury to discuss possible host teams.*

Intitulé du programme de recherche: Trustworthiness and meta-programming for interactive theorem provers
*Title of research program*

**Taille maximum de cette partie : 3 pages (CRCN, ISFP et DR2)**

***Maximum size for this part: 3 pages (CRCN, ISFP and DR2)***

. . . . . .

## Trustworthiness and meta-programming for interactive theorem provers

Interactive theorem provers (ITPs) such as Coq, Lean, Agda, F*, HOL4, Isabelle/HOL, etc. have been successfully used for the verification of (security-)critical software infrastructure, such as the CompCert C compiler [10], the P-256 elliptic curve as a part of Google's BoringSSL library [5], smart contract language interpreters [19], the CakeML compiler for a dialect of Standard ML [9], or the sel4 microkernel [8]. Crucial for the trust in these programs is that the used ITP is trustworthy. However, ITPs are complex programs with little precise specification and opaque implementations – meaning it is hard to ensure they are sound and even harder to continuously argue their soundness during development over many years. Simultaneously, formalising proofs is still an ardeous and time-consuming process – leading to, amongst other things, a high maintenance cost.

Through my work, I want to contribute to improving the trustworthiness *of* ITPs and the maintainability of proofs checked *in* ITPs. Aligning with the lines of work described in the previous form, I am describing three particular lines of work I envision, that I can work on in collaboration with local and international colleagues, as well as future Phd, Master's, and Bachelor's students.

First, I want to work towards a shared platform for verified extraction from various ITPs such as Coq, Lean, Agda, etc. using various back-ends such as OCaml, C, WASM, etc. This line would extend my current work on verifying Coq's extraction to OCaml significantly, and widen the applicability. It would contribute to a higher trustworthiness of ITPs regarding programs extracted from them, while minimising the individual maintenance cost by providing a centralised platform.

Secondly, I want to work on meta-programming techniques for proofs. In particular, I want to develop MetaCoq into a framework for accessible and powerful meta-programming, with the goal to obtain robust tools for modular proofs and programs based on the Coq á la Carte approach co-proposed with Kathrin Stark.

Thirdly, I want to continue formalising mathematics in ITPs and extend my work on synthetic computability theory to cover a graduate curriculum on computability theory and work on the approach to be accessible for theoretical computer science practitioners in their published work, as well as to be used in teaching. Related to this is also an analysis of the computational content of results in computability theory and their constructive status, which can be discovered through an analysis via constructive reverse mathematics.

### 1. A platform for verified extraction of programs from interactive theorem provers

Many of the programs successfully verified in ITPs mentioned above rely on *extraction*, where the program is implemented in the programming language of the ITP to ease verification, and then automatically translated to an industrial programming language such as OCaml, other dialects of the ML family, Haskell, or C. Extracting the verified software to such multi-purpose industrial languages is crucial to interact with unverified software components and for efficient execution.

The guarantees an implementation, verification, and extraction of security-critical software in ITPs provides then crucially rely on the trustworthiness of the tools involved: Both the proof checking in the used ITP and the extraction process have to be trusted, i.e. their implementations are part of the Trusted Computing Base (TCB), which in general should be tried to be kept as small as possible.

Currently, most ITPs support extraction to one major programming language. E.g. Coq can extract to OCaml, Agda to Haskell, HOL4 and Isabelle/HOL to CakeML, Lean 3 to C, F* to OCaml, WASM, and others, etc. If the programming language chosen to implement non-verified software components is fixed, it can then dictate the ITP to use for the verified components – a highly unsatisfying situation.

Furthermore, extraction processes are usually argued correct on paper for idealised systems, but their implementation itself is rarely a trustworthy program verified in an ITP. However, the actual implementations of extraction are non-trivial programs subject to bugs, as can be seen by long-standing open issues in e.g. Coq's extraction process[2]. Besides that, extraction often supports on-the-fly optimisations to make the resulting code more readable or more efficient. Once again in the case of Coq, these optimisations are indeed crucial, but come only with empirical correctness guarantees and not even a specification of correctness on paper.

I want to work towards a unified platform for efficient and correct extraction from ITPs to industrial programming languages, where the intermediate language, optimisations, and compilation to backends are verified in the Coq proof assistant, the leading ITP for the verification of software.

The proposed project will both reduce the TCB of verified programs extracted to industrial programming languages and enable extraction from various ITPs to various programming languages. Central to the platform is an intermediate language which makes it easy to connect to existing ITPs, for which we propose the $\lambda_\square$ language underlying extraction from the Coq proof assistant [57], which we already formalised in Coq in previous work [63]. Choosing $\lambda_\square$ has several advantages. The first is mathematical: It is essentially an untyped, operational specification of the programming language underlying the Coq proof assistant. Since the underlying theory of Coq is relatively expressive compared to the ones of other type-theoretic ITPs in terms of definable programs, translating programs from Lean and Agda to this language will be more straightforward than any other possible direction. The second advantage is on the proof engineering side: In previous projects we have amassed substantial machine-checked theory for $\lambda_\square$, among other results a translation into C in the CertiCoq project [1], strengthening the conjecture that $\lambda_\square$ is particularly suitable for machine-checked proofs.

[2] https://github.com/coq/coq/issues/6614

For each ITP, the connection between the theory underlying the ITP and the intermediate language is established in the ITP itself. Specification of an ITP in itself rather than another one is crucial to ease keeping it up-to-date for developers of the ITP. E.g. for Coq, we use the specification of Coq in Coq as provided by the MetaCoq project [14]. For other ITPs, such self-formalisations will first have to be carried out. For every ITP involved, we will then just have to port the specification of $\lambda_\square$ from Coq. In general, porting theories from one ITP to another is highly complex, thus choosing the specification of $\lambda_\square$ creates a good and stable interface because it is compact, does not change, and the correctness of the porting can even be manually audited.
After that, all further verification of e.g. optimisations can then happen just on the intermediate language in the Coq proof assistant, simplifying the overall architecture and removing the need for more manual audits.
This proposed project would go significantly beyond state of the art of machine-checked correctness of programs. While projects connecting different ITPs for proofs exist, e.g. the Dedukti project, a shared basis to obtain executable programs from different ITPs is not available and every ITP implements a conceptually similar process subject to potential problems. Individually, the extraction process of HOL4 and Isabelle/HOL to CakeML are verified [53, 60], and we are working on verifying parts of Coq's extraction to Ocaml [49]. However, by having a shared intermediate language, both extraction from ITPs into this language and compilation into industrial programming languages can be verified once and for all, removing the extraction process from the TCB of all of these ITPs entirely. Furthermore, it enables new target languages for program extraction from ITPs without adapting the code base of the ITP itself, and it allows the specification and verification of optimisations with shared use.
In the mid-term, I envision to connect at least the ITPs Coq, Lean, Agda, and F* as source, and OCaml, CakeML, and C as targets, with the long-term goal to implement backends for Haskell and WASM. In the past, such a project would not have been feasible, but with the maturity of the MetaCoq project formalising Coq in Coq [14], the Lean prover being implemented in Lean itself [12], the Agda core project [3] which is in the process of formally specifying Agda, and previous machine-checked formalisations of the theory of F*, a formal specification of all of these sources is in reach. We recently devised a formal specification of the intermediate language of the OCaml compiler from an interpreter [24], which can serve as OCaml backend. In the short-term, I aim at porting the existing formal specification of CakeML to Coq, use the formal specification of C from [10] as used in the CertiCoq project [1], and possibly use the formalisation of WASM [16].

### 2. Verifiable Meta-programming and Modularity

Meta-programming techniques for proofs are still an underdeveloped area of research. In the mid-term, I want to develop MetaCoq's meta-programming facilities to a robust framework, and apply it to improve the situation regarding three problems: modularity of proofs, maintainabilibity of proofs, and portability of proofs between ITPs.
First, although several solutions how to modularly develop proofs and verified programs in ITPs have been suggested [23, 62, 54, 59, 20], none of them has been adopted by the interactive theorem proving community. With Kathrin Stark (Heriot Watt University) and Kenji Maillard (Inria Nantes), I want to continue working on our mathematically light-weight proposal for modular proofs [50], taking its power from clearly specified minimal automation tools based on meta-programming. The approach can deal well with non-dependent inductive types, but does not support dependent types or modularity in the type of predicates, both crucial to scale to realistic developments in the Coq ecosystem. We intend to mature the approach to be able to help in large developments on the short term.
Secondly, repairing existing proofs after definitions have been slightly changed and extended consumes significant proof engineering time. With the use of ITPs rising in the general mathematical community – a hope that seems realistic following the recent adaption of Lean and the success of e.g. the liquid tensor experiment – such questions become of even higher importance: If maintaining proofs with new versions of the ITP software becomes a serious burden or impossible, use of ITPs will fade quickly again. Meta-programming tools to repair proofs can help in mitigating this danger.
Third, the ecosystem of ITPs is highly diverse: While it is difficult to translate statements from one ITP to another, translating proofs can actually be impossible due to differences in the foundational theory. As a short-term goal, I intend to work on such translations for Lean and Coq with Sebastian Ullrich (KIT), employing meta-programming techniques on both sides.

### 3. Synthetic Computability and Constructive Reverse Mathematics

Computability theory is widely taught in almost every undergraduate computer science curriculum and a central ingredient of meta-mathematical aspects of the foundations of logic, e.g. playing a central role in Gödel's completeness and incompleteness results. However, the style of presentation and mathematical development of content have not changed significantly in the last decades. Moreover, computability theory was until recently one of the last areas of mathematics not applicable to machine-checked proofs, because informal applications of the Church Turing thesis (stating that intuitively computable functions are computable in a model of computation) cannot be easily translated to formal proofs.
In my PhD work, I have developed a synthetic approach to computability theory, where the functions definable in the Coq proof assistant can be seen as the model of computation. Synthetic approaches to computability theory have been known since the 1980s, but used to be incompatible with classical logic (i.e. the law of excluded middle is refutable). By basing the theory on top of CIC, the type-theoretic foundation underlying the Coq proof assistant, this burden is lifted. Consequently, computability theory can be carried out in a proof assistant, and sufficient expert knowledge of either the

proof and computability to do meaningful, interesting work can be amassed in a reasonable time, as witnessed by the various Bachelor's theses I have advised on several topics.
The most challenging open questions are how to explain oracle computation in this setting and how to define models of type theory flexible enough to show the approach consistent with other widely assumed axioms. The universally present question is the status of computability theory results in (constructive) reverse mathematics where I intend to collaborate further with Dominik Kirst and Gert Smolka (Uni Saarland). I also intend to maintain and foster more collaborations with the French school of classical realizability, for example with Hugo Herbelin (Inria) and Etienne Miquey (U. of Marseille), and the realizability community, for example with Andrej Bauer (U. of Ljubljana).
As a result of this line and mid-term goal, I hope to obtain a machine-checked theory of a graduate curriculum of computability, which can ultimately contribute back to teaching these foundational topics to students, but also already in short-term to serve as foundation to enable researchers to present their (un)decidability and (un)computability results indisputably formalised in an ITP.

## 4. Integration with Cambium

The Cambium team is an ideal team for my lines of work. Given that the meta-theory of programming languages is a central topic of the team, I expect to be able to work on my research goal of modular reasoning in dependent type theory with almost all members. Furthermore, Cambium is one of the teams with the most amassed expertise of the OCaml programming language.
I would like to work with François Pottier on a semantic typing relation for OCaml, which will crucially be needed in the project on verified extraction. His work would also intersect particularly well with my work on modularity for proofs, e.g. in the work on logical relations or on the Mezzo programming language. Didier Rémy is working on the semantics of modules in OCaml, and I would like to also specify the semantics of modules in Coq formally as part of the MetaCoq project, allowing to talk about verified extraction of Coq modules to OCaml modules. Furthermore, I would like to work with Xavier Leroy on verifying parts of the OCaml toolchain, such as optimisations in the middle-end, but also on verifying optimisations of Coq's extraction to OCaml and more generally optimisations in the envisioned extraction platform. Luc Maranget is an expert in pattern-matching, which could lead to work on verifying OCaml's pattern-matching compilation in the front-end in Coq, while Jean-Marie Madiot could become a collaborator on a C back-end of the extraction platform.

## References


[1] A. Anand, A. Appel, G. Morrisett, Z. Paraskevopoulou, R. Pollack, O. S. Belanger, M. Sozeau, and M. Weaver. Certicoq: A verified compiler for coq. In *Workshop on Coq for Programming Languages (CoqPL)*, 2017.

[2] B. Bernardo, R. Cauderlier, Z. Hu, B. Pesin, and J. Tesson. Mi-cho-coq, a framework for certifying tezos smart contracts. In *Formal Methods. FM 2019 International Workshops*, pages 368–379, Cham, 2020. Springer.

[3] J. Cockx. Agda core: A trustworthy and extensible core language for agda. In *NWO Veni Grant*, 2020.

[4] S. Dolan. Malfunctional programming. In *ML Workshop*, 2016.

[5] A. Erbsen, J. Philipoom, J. Gross, R. Sloan, and A. Chlipala. Simple high-level code for cryptographic arithmetic-with proofs, without compromises. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1202–1219. IEEE, 2019.

[6] Y. Forster, M. Sozeau, P. Giraud, P.-M. Pédrot, and N. Tabareau. Extraction to ocaml from coq: Operational correctness verified in coq. In *ML family workshop*, 2022.

[7] L. Hupel and T. Nipkow. A verified compiler from Isabelle/HOL to CakeML. In *ESOP 2018*, pages 999–1026. Springer, 2018.

[8] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, et al. sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 207–220, 2009.

[9] R. Kumar, M. O. Myreen, M. Norrish, and S. Owens. Cakeml: a verified implementation of ML. In *ACM SIGPLAN Notices*, volume 49, pages 179–191. ACM, 2014.

[10] X. Leroy, S. Blazy, D. Kästner, B. Schommer, M. Pister, and C. Ferdinand. CompCert - A Formally Verified Optimizing Compiler. In *ERTS 2016: Embedded Real Time Software and Systems, 8th European Congress*.

[11] P. Letouzey. *Programmation fonctionnelle certifiée: l'extraction de programmes dans l'assistant Coq*. PhD thesis, 2004.

[12] L. d. Moura and S. Ullrich. The lean 4 theorem prover and programming language. In *Automated Deduction–CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings 28*, pages 625–635. Springer, 2021.

[13] M. O. Myreen and S. Owens. Proof-producing translation of higher-order logic into pure and stateful ML. *Journal of Functional Programming*, 24(2-3):284–315, 2014.

[14] M. Sozeau, A. Anand, S. Boulier, C. Cohen, Y. Forster, F. Kunze, G. Malecha, N. Tabareau, and T. Winterhalter. The MetaCoq Project. *Journal of Automated Reasoning*, Feb. 2020.

[15] M. Sozeau, S. Boulier, Y. Forster, N. Tabareau, and T. Winterhalter. Coq coq correct! verification of type checking and erasure for coq, in coq. *Proc. ACM Program. Lang.*, 4(POPL), Dec. 2019.

[16] C. Watt, X. Rao, J. Pichon-Pharabod, M. Bodin, and P. Gardner. Two mechanisations of webassembly 1.0. In *Formal Methods: 24th International Symposium, FM 2021, Virtual Event, November 20–26, 2021, Proceedings 24*, pages 61–79. Springer, 2021.

## Formulaire 5 — LISTE COMPLÈTE DES CONTRIBUTIONS
*Form 5 — COMPLETE LIST OF CONTRIBUTIONS*

All peer-reviewed contributions are available on my website at https://yforster.de. According to Google Scholar, my papers have been cited 762 times.

### 1. Publications caractéristiques/*Representative publications*

- Matthieu Sozeau, Simon Boulier, Yannick Forster, Nicolas Tabareau, Théo Winterhalter. "Coq Coq correct! verification of type checking and erasure for Coq, in Coq" *PACMPL 4(POPL)*: 8:1-8:28 (2020).
- Yannick Forster, Kathrin Stark. "Coq à la carte: a practical approach to modular syntax with binders". *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020.
- Yannick Forster. "Parametric Church's Thesis: Synthetic Computability Without Choice", *Symposium on Logical Foundations of Computer Science*, LFCS 2022. *won the Rosser prize for the best paper by junior researchers*

### 2. Politique de publication/*Publication policy*

### 3. Publications

I try to pick conferences thematically suitable for the work, with an expert PC in the hope of receiving helpful feedback for publication.
The author order on papers is usually alphabetic. Exceptions are [J4] / [C8], where Dominique Larchey-Wendling initiated the project and wrote significantly more than half of the Coq code, [J3] / [C15], where the MetaCoq project coordinator was assigned as first author, and [C4] and [C13] where I was more in the role of an advisor for the whole project.

#### 3.1 Revues internationales/*International journals*

[J5] Yannick Forster, Dominik Kirst, Dominik Wehr. "Completeness Theorems for First-Order Logic Analysed in Constructive Type Theory (extended version)". Journal of Logic and Computation. Extended version of [C10].

[J4] Dominique Larchey-Wendling and Yannick Forster. "Hilbert's Tenth Problem in Coq (extended version)". Logical Methods in Computer Science (LMCS). Extended version pf [C8].

[J3] Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. "The MetaCoq Project." *Journal of Automated Reasoning*, JAR 2020.

[J2] Yannick Forster, Ohad Kammar, Sam Lindley, Matija Pretnar. "On the expressive power of user-defined effects: effect handlers, monadic reflection, delimited control". *Journal of Functional Programming*, JFP 2019. Extended version of [C2].

[J1] Yannick Forster and Gert Smolka. "Call-by-Value Lambda Calculus as a Model of Computation in Coq". *Journal of Automated Reasoning*, JAR 2018. Extended version of [C1].

#### 3.2 Conférence internationales avec comité de lecture/*Reviewed international conferences*

[C21] Yannick Forster, Felix Jahn, Gert Smolka. "A Computational Cantor-Bernstein and Myhill's Isomorphism Theorem in Constructive Type Theory", *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2023.

[C20] Yannick Forster, Felix Jahn. "Constructive and Synthetic Reducibility Degrees: Post's Problem for Many-one and Truth-table Reducibility in Coq", *31st Conference for Computer Science Logic*, CSL 2023.

[C19] Yannick Forster, Nils Lauermann, Fabian Kunze. "Synthetic Kolmogorov Complexity in Coq", *International Conference on Interactive Theorem Proving*, ITP 2022.

[C18] Yannick Forster. "Parametric Church's Thesis: Synthetic Computability Without Choice", *Symposium on Logical Foundations of Computer Science*, LFCS 2022.

[C17] Yannick Forster, Fabian Kunze, Gert Smolka, Maximilian Wuttke. "A Mechanised Proof of the Time Invariance Thesis for the Weak Call-By-Value $\lambda$-Calculus". *International Conference on Interactive Theorem Proving*, ITP 2021.

[C16] Yannick Forster. “Church’s thesis and related axioms in Coq’s type theory” *29th Conference for Computer Science Logic*, CSL 2021.

[C15] Matthieu Sozeau, Simon Boulier, Yannick Forster, Nicolas Tabareau, Théo Winterhalter. “Coq Coq correct! verification of type checking and erasure for Coq, in Coq” *PACMPL 4(POPL)*: 8:1-8:28 (2020).

[C14] Yannick Forster, Fabian Kunze, Maximilian Wuttke. “Verified programming of Turing machines in Coq”. *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020.

[C13] Simon Spies, Yannick Forster. “Undecidability of higher-order unification formalised in Coq”. *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020.

[C12] Yannick Forster, Kathrin Stark. “Coq à la carte: a practical approach to modular syntax with binders”. *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020.

[C11] Yannick Forster, Fabian Kunze, Marc Roth. “The weak call-by-value $\lambda$-calculus is reasonable for both time and space”. *PACMPL 4 (POPL)*: 27:1-27:23 (2020).

[C10] Yannick Forster, Dominik Kirst, Dominik Wehr. “Completeness Theorems for First-Order Logic Analysed in Constructive Type Theory”. *Symposium on Logical Foundations of Computer Science*, LFCS 2020.

[C9] Yannick Forster and Fabian Kunze. “A certifying extraction with time bounds from Coq to call-by-value $\lambda$-calculus”. *International Conference on Interactive Theorem Proving*, ITP 2019.

[C8] Dominique Larchey-Wendling and Yannick Forster. “Hilbert’s Tenth Problem in Coq”. *4th International Conference on Formal Structures for Computation and Deduction*, FSCD 2019.

[C7] Yannick Forster, Steven Schäfer, Simon Spies, Kathrin Stark. “Call-By-Push-Value in Coq: Operational, Equational and Denotational Theory”. *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2019.

[C6] Yannick Forster, Dominik Kirst, Gert Smolka. “On Synthetic Undecidability in Coq, with an Application to the Entscheidungsproblem”. *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2019.

[C5] Yannick Forster and Dominique Larchey-Wendling. “Certified Undecidability of Intuitionistic Linear Logic via Binary Stack Machines and Minsky Machines”. *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2019.

[C4] Fabian Kunze, Gert Smolka, Yannick Forster. “Formal Small-step Verification of a Call-by-value Lambda Calculus Machine”. *Asian Symposium on Programming Languages and Systems*, APLAS 2018.

[C3] Yannick Forster, Edith Heiter, Gert Smolka. “Verification of PCP-Related Computational Reductions in Coq”. *International Conference on Interactive Theorem Proving*, ITP 2018.

[C2] Yannick Forster, Ohad Kammar, Sam Lindley, Matija Pretnar. “On the expressive power of user-defined effects: Effect handlers, monadic reflection, delimited control.” *Proceedings of the ACM on Programming Languages 1*. ICFP 2017.

[C1] Yannick Forster and Gert Smolka. “Weak Call-by-Value Lambda Calculus as a Model of Computation in Coq”. *International Conference on Interactive Theorem Proving*, ITP 2017.

### 3.3 Livres et chapitres de livre/*Books and book chapters*

### 3.4 Autres publications internationales (posters, articles courts)/*Other international publications (posters, short papers)*

[W15] Yannick Forster, Dominik Kirst. “Synthetic Turing Reducibility in CIC”, *28th International Conference on Types for Proofs and Programs*, TYPES 2022.

[W15] Dominik Kirst, Niklas Mück, Yannick Forster. “Synthetic Versions of the Kleene-Post and Post’s Theorem”, *28th International Conference on Types for Proofs and Programs*, TYPES 2022.

[W14] Yannick Forster, Felix Jahn, Gert Smolka. “Myhill Isomorphism Theorem and a Computational Cantor-Bernstein Theorem in Constructive Type Theory”, *28th International Conference on Types for Proofs and Programs*, TYPES 2022. Published as [C21].

[W13] Yannick Forster, Matthieu Sozeau. “Aspects of a machine-checked intermediate language for extraction from Coq, in MetaCoq” *28th International Conference on Types for Proofs and Programs*, TYPES 2022.

[W12] Dominik Kirst, Johannes Hostert, Andrej Dudenhefner, Yannick Forster, Marc Hermes, Mark Koch, Dominique Larchey-Wendling, Niklas Mück, Benjamin Peters, Gert Smolka, Dominik Wehr, “A Coq Library for Mechanised First-Order Logic”. *Coq Workshop 2022*, Haifa, Israel.

[W11] Yannick Forster, Matthieu Sozeau, Pierre Giraud, Pierre-Marie Pédrot, Nicolas Tabareau. “Extraction to OCaml from Coq: Operational Correctness Verified in Coq”. *ML family workshop 2022*, Ljubljana, Slovenia.

[W10] Matthieu Sozeau, Meven Lennon-Bertrand, Yannick Forster. “The Curious Case of Case: correct and efficient case representation in Coq and MetaCoq”. *The first Workshop on the Implementation of Type Systems (WITS 2022)*, online, 2022.

[W9] Bohdan Liesnikov, Marcel Ullrich, Yannick Forster. “Generating induction principles and subterm relations for inductive types using MetaCoq”. *Coq Workshop 2020*, online.

[W8] Yannick Forster, Dominik Kirst, Florian Steinberg. “Towards Extraction of Continuity Moduli in Coq” *26th International Conference on Types for Proofs and Programs*, TYPES 2020.

[W7] Yannick Forster, Dominique Larchey-Wendling, Andrej Dudenhefner, Edith Heiter, Dominik Kirst, Fabian Kunze, Gert Smolka, Simon Spies, Dominik Wehr, Maximilian Wuttke. “A Coq Library of Undecidable Problems”. *The Sixth International Workshop on Coq for Programming Languages (CoqPL 2020)*, New Orleans, USA 2020.

[W6] Matthieu Sozeau, Yannick Forster, Simon Boulier, Nicolas Tabareau and Théo Winterhalter. “Coq Coq Codet! - Towards a Verified Toolchain for Coq in MetaCoq”. Coq Workshop 2019, Portland, USA. Published as [C15].

[W5] Yannick Forster and Matthieu Sozeau. “Mechanically verified type and proof erasure for Coq ”. Facets of Realizability workshop 2019, Paris, France. Published as [C15].

[W4] Yannick Forster and Dominique Larchey-Wendling. “A constructive Coq-library for the mechanisation of undecidability”. MLA workshop 2019, Nancy, France. Published as [C5].

[W3] Yannick Forster and Dominique Larchey-Wendling. “Towards a library of formalised undecidable problems in Coq: The undecidability of intuitionistic linear logic”. *Syntax and Semantics of Low-Level Languages workshop*, LOLA 2018, Oxford, UK. Published as [C5].

[W2] Yannick Forster, Fabian Kunze, Marc Roth. “The strong invariance thesis for a lambda-calculus”. *Syntax and Semantics of Low-Level Languages workshop*, LOLA 2017, Reykjavik, Iceland. Published as [C11].

[W1] Yannick Forster and Fabian Kunze. “Verified Extraction from Coq to a Lambda-Calculus”. *Coq Workshop 2016*, Nancy, France. Published as [C9].

### 3.5 Revues nationales/*National journals*

### 3.6 Conférence nationales avec comité de lecture/*Reviewed national conferences*

### 3.7 Rapports de recherche et articles soumis/*Research reports and publications under review*

## 4. Développements technologiques : logiciel ou autre réalisation / *Technology development : software or other realization*

- coq-library-undecidability: `Family=research; Audience=community; evolution=lts; Duration=5; contribution=leader; Url=https://github.com/uds-psl/coq-library-undecidability` The Coq Library of Undecidability Proofs contains mechanised reductions to establish undecidability results in Coq. It is a collaborative project of more than 16 contributors, spanning more than 100.000 lines of Coq code. The code of papers [32, 38, 45, 64, 44, 43, 25, 55, 27] is all contributed and centrally maintained there.

- coq-synthetic-computability: `Family=research; Audience=community; evolution=lts; Duration=3; contribution=leader; Url=https://github.com/uds-psl/coq-synthetic-computability` A formalisation of synthetic computability in Coq, spanning around 20.000 lines of results from computability theory based on the axioms “Church’s thesis”. Contains the code of [33, 41, 34].

- coq-metacoq-erasure: `Family=research; Audience=community; evolution=lts; Duration=4; contribution=leader; Url=https://github.com/MetaCoq/metacoq/tree/coq-8.16/erasure, https://github.com/yforster/coq-malfunction/` The verification of type and proof erasure from MetaCoq’s intermediate calculus PCUIC to $\lambda_{\square}$, several verified transformations bringing the semantics of $\lambda_{\square}$ closer to that of OCaml, and a verified translation from $\lambda_{\square}$ to Lambda, the intermediate language of the OCaml compiler. Contains code corresponding to papers [63, 49].

- CertiCoq: `Family=research; Audience=community; evolution=lts; Duration=3; contribution=devel; Url=https://github.com/CertiCoq/certicoq` A verified compiler from Coq to C.

Furthermore:

- coq-library-complexity: `Family=research; Audience=community; evolution=lts; Duration=4; contribution=instigator,devel; Url=https://github.com/uds-psl/coq-library-complexity` Project lead by Fabian Kunze on formalising results from complexity theory in Coq, where I was involved in the creation through the time and space equivalence proof of the weak call-by-value $\lambda$-calculus [42] and the creation of tactics allowing the automatically extract Coq functions to $\lambda$ terms [19].
- coq-library-fol: `Family=research; Audience=partners; evolution=lts; Duration=1; contribution=instigator,devel; Url=https://github.com/uds-psl/coq-library-fol` Project lead by Dominik Kirst on formalising results from meta-mathematics and first-order logic, where I was involved in the creation through the constructive analysis of the completeness proof [39, 40].
- cbpv-in-coq `Family=research; Audience=partners; evolution=nofuture; Duration=1; contribution=leader; Url=https://www.ps.uni-saarland.de/extras/cbpv-in-coq/` Formalisation accompanying [47].
- Autosubst 2 `Family=research; Audience=community; evolution=lts; Duration=3; contribution=softcont; Url=https://github.com/uds-psl/autosubst2` A tool to generate boilerplate for syntax and binding, where I collaborated to integrate support for modular syntax [51], and help maintain the repository, CI, and documentation.

## 5. Impact socio-économique et transfert / *Socio-economic impact and transfer*

Our verified extraction is expected to be industrially used by Formal Vindications (`https://formalv.com/`).

[17] D. Annenkov, J. B. Nielsen, and B. Spitters. Concert: A smart contract certification framework in coq. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 215–228, 2020.

[18] A. Bauer. First steps in synthetic computability theory. *Electronic Notes in Theoretical Computer Science*, 155:5–31, 2006.

[19] B. Bernardo, R. Cauderlier, Z. Hu, B. Pesin, and J. Tesson. Mi-cho-coq, a framework for certifying tezos smart contracts. In *Formal Methods. FM 2019 International Workshops*, pages 368–379, Cham, 2020. Springer.

[20] O. Boite. Proof reuse with extended inductive types. In *International Conference on Theorem Proving in Higher Order Logics*, pages 50–65. Springer, 2004.

[21] J. Cockx, O. Melkonian, L. Escot, J. Chapman, and U. Norell. Reasonable agda is correct haskell: writing verified haskell using agda2hs. In *Proceedings of the 15th ACM SIGPLAN International Haskell Symposium*, pages 108–122, 2022.

[22] T. Cogumbreiro and Y. Forster. Towards a mechanized theory of computation for education. In *28th International Conference on Types for Proofs and Programs (TYPES 2022)*, 2022.

[23] B. Delaware, B. C. d S Oliveira, and T. Schrijvers. Meta-theory à la carte. In *ACM SIGPLAN Notices*, volume 48, pages 207–218. ACM, 2013.

[24] S. Dolan. Malfunctional programming. In *ML Workshop*, 2016.

[25] A. Dudenhefner. The undecidability of system F typability and type checking for reductionists. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–10. IEEE, 2021.

[26] A. Dudenhefner. Certified decision procedures for two-counter machines. In A. P. Felty, editor, *7th International Conference on Formal Structures for Computation and Deduction, FSCD 2022, August 2-5, 2022, Haifa, Israel*, volume 228 of *LIPIcs*, pages 16:1–16:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[27] A. Dudenhefner. Constructive many-one reduction from the halting problem to semi-unification. In F. Manea and A. Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, pages 18:1–18:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[28] Y. Forster. Church's thesis and related axioms in coq's type theory, 2020.

[29] Y. Forster. Church's thesis and related axioms in Coq's type theory. In C. Baier and J. Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25-28, 2021, Ljubljana, Slovenia (Virtual Conference)*, volume 183 of *LIPIcs*, pages 21:1–21:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[30] Y. Forster. *Computability in Constructive Type Theory*. PhD thesis, Saarland University, 2021.

[31] Y. Forster. Parametric church's thesis: Synthetic computability without choice. In S. Artemov and A. Nerode, editors, *Logical Foundations of Computer Science*, pages 70–89, Cham, 2022. Springer International Publishing.

[32] Y. Forster, E. Heiter, and G. Smolka. Verification of pcp-related computational reductions in Coq. In J. Avigad and A. Mahboubi, editors, *Interactive Theorem Proving - 9th International Conference, ITP 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10895 of *Lecture Notes in Computer Science*, pages 253–269. Springer, 2018.

[33] Y. Forster and F. Jahn. Constructive and synthetic reducibility degrees: Post's problem for many-one and truth-table reducibility in Coq. In *31st EACSL Annual Conference on Computer Science Logic, CSL 2023, February 13-16, 2023, Warsaw, Poland*, volume 252 of *LIPIcs*, pages 16:1–16:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[34] Y. Forster, F. Jahn, and G. Smolka. A computational cantor-bernstein and myhill's isomorphism theorem in constructive type theory (proof pearl). In R. Krebbers, D. Traytel, B. Pientka, and S. Zdancewic, editors, *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2023, Boston, MA, USA, January 16-17, 2023*, pages 159–166. ACM, 2023.

[35] Y. Forster, O. Kammar, S. Lindley, and M. Pretnar. On the expressive power of user-defined effects: effect handlers, monadic reflection, delimited control. *Proc. ACM Program. Lang.*, 1(ICFP):13:1–13:29, 2017.

[36] Y. Forster, O. Kammar, S. Lindley, and M. Pretnar. On the expressive power of user-defined effects: Effect handlers, monadic reflection, delimited control. *J. Funct. Program.*, 29:e15, 2019.

[37] Y. Forster, D. Kirst, and G. Smolka. On synthetic undecidability in coq, with an application to the entscheidungsproblem. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 38–51, 2019.

[38] Y. Forster, D. Kirst, and G. Smolka. On synthetic undecidability in Coq, with an application to the Entscheidungsproblem. In A. Mahboubi and M. O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, pages 38–51. ACM, 2019.

[39] Y. Forster, D. Kirst, and D. Wehr. Completeness theorems for first-order logic analysed in constructive type theory. In S. N. Artëmov and A. Nerode, editors, *Logical Foundations of Computer Science - International Symposium, LFCS 2020, Deerfield Beach, FL, USA, January 4-7, 2020, Proceedings*, volume 11972 of *Lecture Notes in Computer Science*, pages 47–74. Springer, 2020.

[40] Y. Forster, D. Kirst, and D. Wehr. Completeness theorems for first-order logic analysed in constructive type theory (extended version). *arXiv preprint arXiv:2006.04399*, 2020.

[41] Y. Forster, F. Kunze, and N. Lauermann. Synthetic Kolmogorov Complexity in Coq. Mar. 2022.

[42] Y. Forster, F. Kunze, and M. Roth. The weak call-by-value $\lambda$-calculus is reasonable for both time and space. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–23, 2019.

[43] Y. Forster, F. Kunze, G. Smolka, and M. Wuttke. A mechanised proof of the time invariance thesis for the weak call-by-value $\lambda$-calculus. In L. Cohen and C. Kaliszyk, editors, *12th International Conference on Interactive Theorem Proving, ITP 2021, June 29 to July 1, 2021, Rome, Italy (Virtual Conference)*, volume 193 of *LIPIcs*, pages 19:1–19:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[44] Y. Forster, F. Kunze, and M. Wuttke. Verified programming of turing machines in Coq. In J. Blanchette and C. Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 114–128. ACM, 2020.

[45] Y. Forster and D. Larchey-Wendling. Certified undecidability of intuitionistic linear logic via binary stack machines and Minsky machines. In A. Mahboubi and M. O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, pages 104–117. ACM, 2019.

[46] Y. Forster, D. Larchey-Wendling, A. Dudenhefner, E. Heiter, D. Kirst, F. Kunze, G. Smolka, S. Spies, D. Wehr, and M. Wuttke. A Coq library of undecidable problems. In *The Sixth International Workshop on Coq for Programming Languages (CoqPL 2020)*., 2020.

[47] Y. Forster, S. Schäfer, S. Spies, and K. Stark. Call-by-push-value in coq: operational, equational, and denotational theory. In A. Mahboubi and M. O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, pages 118–131. ACM, 2019.

[48] Y. Forster and G. Smolka. Weak call-by-value lambda calculus as a model of computation in coq. In *International Conference on Interactive Theorem Proving*, pages 189–206. Springer, 2017.

[49] Y. Forster, M. Sozeau, P. Giraud, P.-M. Pédrot, and N. Tabareau. Extraction to ocaml from coq: Operational correctness verified in coq. In *ML family workshop*, 2022.

[50] Y. Forster and K. Stark. Coq à la carte: a practical approach to modular syntax with binders. In J. Blanchette and C. Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 186–200. ACM, 2020.

[51] Y. Forster and K. Stark. Coq à la carte: a practical approach to modular syntax with binders. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 186–200, 2020.

[52] J. Hostert, A. Dudenhefner, and D. Kirst. Undecidability of dyadic first-order logic in coq. In J. Andronick and L. de Moura, editors, *13th International Conference on Interactive Theorem Proving, ITP 2022, August 7-10, 2022, Haifa, Israel*, volume 237 of *LIPIcs*, pages 19:1–19:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[53] L. Hupel and T. Nipkow. A verified compiler from Isabelle/HOL to CakeML. In *ESOP 2018*, pages 999–1026. Springer, 2018.

[54] S. Keuchel and T. Schrijvers. Generic datatypes à la carte. In *Proceedings of the 9th ACM SIGPLAN Workshop on Generic Programming*, pages 13–24. ACM, 2013.

[55] D. Kirst and D. Larchey-Wendling. Trakhtenbrot's theorem in coq: Finite model theory through the constructive lens. *Log. Methods Comput. Sci.*, 18(2), 2022.

[56] D. Larchey-Wendling. Synthetic undecidability of MSELL via FRACTRAN mechanised in coq. In N. Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference)*, volume 195 of *LIPIcs*, pages 18:1–18:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[57] P. Letouzey. *Programmation fonctionnelle certifiée: l'extraction de programmes dans l'assistant Coq*. PhD thesis, 2004.

[58] B. Liesnikov, M. Ullrich, and Y. Forster. Generating induction principles and subterm relations for inductive types using metacoq. *Coq Workshop 2020*, 2020.

[59] A. Mulhern. Proof weaving. In *Proceedings of the First Informal ACM SIGPLAN Workshop on Mechanizing Metatheory*, 2006.

[60] M. O. Myreen and S. Owens. Proof-producing translation of higher-order logic into pure and stateful ML. *Journal of Functional Programming*, 24(2-3):284–315, 2014.

[61] F. Richman. Church's thesis without tears. *The Journal of symbolic logic*, 48(3):797–803, 1983.

[62] C. Schwaab and J. G. Siek. Modular type-safety proofs in agda. In *Proceedings of the 7th workshop on Programming languages meets program verification*, pages 3–12. ACM, 2013.

[63] M. Sozeau, S. Boulier, Y. Forster, N. Tabareau, and T. Winterhalter. Coq coq correct! verification of type checking and erasure for coq, in coq. *Proc. ACM Program. Lang.*, 4(POPL), Dec. 2019.

[64] S. Spies and Y. Forster. Undecidability of higher-order unification formalised in Coq. In J. Blanchette and C. Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 143–157. ACM, 2020.

[65] D. Larchey-Wendling and Y. Forster. Hilbert's tenth problem in Coq. In H. Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 27:1–27:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[66] M. Sozeau, A. Anand, S. Boulier, C. Cohen, Y. Forster, F. Kunze, G. Malecha, N. Tabareau, and T. Winterhalter. The MetaCoq project. *J. Autom. Reason.*, 64(5):947–999, 2020.

[67] M. Sozeau, S. Boulier, Y. Forster, N. Tabareau, and T. Winterhalter. Coq Coq Correct! verification of type checking and erasure for Coq, in Coq. *Proc. ACM Program. Lang.*, 4(POPL):8:1–8:28, 2020.