Accompagnement au Diplôme des Normaliens

2025 / 2026

Arthur ADJEDJ

Types d'activités

S	Gérée par la structure d'appartenance
T	Activité transversale
?	Non proposée cette année

Etats associés

I	Demande d'inscription
A	Inscription acceptée
R	Inscription refusée
V	Validé
N	Non validé
F	Validation forcée
D	Dispense
E	Equivalence

Les activités du diplôme

ADJEDJ Arthur

Structure actuellement associée au diplôme : INFO

Tronc commun

- S TC1 - L3 renforcé I A V
- S TC2 - M1 renforcé I A V
- S TC3 - M2R I A
- T TC4 - Conférences du diplôme I A V

Anglais professionnel

- T TC5_1 - Cambridge I A V
- T TC5_2 - IELTS
- T TC6 - SWAP I A V
- S TC7 - Première expérience de reche... I A V
- T TC8 - Expérience engagement nor...
- T TC9 - TEDS

Années spécifiques parcours

- S ES1 - M2 FESup
- S R1 - ARPE I A V
- S P1 - Interface I A N
- T RT1 - ARIA
- T RT2 - ARTeQ
- T RT3 - ARRC
- T RT4 - ARéco
- T RT5 - ASPEN
- S R1B - ARPF

Enseignement supérieur

- T ES2 - Activités d'enseignement
- T ES3 - Tutorat
- T ES4 - Médiation et vulgarisation sci... I A V

Recherche

- S R2 - Deuxième expérience de reche... I A V
- S R3 - Article soumis ou conférence d... I A V
- S R4 - UE analyse d'articles
- T RT1 - ARIA
- T RT2 - ARTeQ
- T RT3 - ARRC
- T RT4 - ARéco

Pluridisciplinarité

- S P1 - Interface I A N
- T P2 - PIC
- S P3 - UE dans une autre discipline I A V
- S P4 - MOOC

UE transversale

- T P5_2 - Initiation à la philosophie et l...
- T P5_8 - Controverses en santé et env...
- T P5_12 - Climat, énergies, environne...
- T P5_13 - Introduction aux géosciences
- T P5_14 - Eléments de sciences cogni...
- T P5_19 - Enjeux numériques du mon...
- T P5_22 - Matériaux anciens et du pat...
- T P5_23 - Les apports de la pédagogi...
- T P5_30 - Penser le vivant aujourd'hui
- T P5_31 - Femmes en sciences/questi...
- T P5_32 - Histoire, sociologie et médi...
- T P5_33 - Introduction aux études de ...
- T P5_1 - Culture scientifique et techni...
- T P5_3 - Valorisation de la recherche

- T P5_4 - Sciences, histoire, sociétés
- T P5_5 - Sciences et technologies esp...
- T P5_6 - Histoire des sciences
- T P5_7 - L'expertise scientifique
- T P5_9 - Sociologie des sciences
- T P5_10 - Histoire environnementale
- T P5_11 - Aux frontières de la recherc...
- T P5_15 - Pratique de l'enseignement
- T P5_16 - Femmes en sciences
- T P5_17 - Ecrire l'histoire des sciences...
- T P5_18 - Introduction à la philosophi...
- T P5_20 - Robots et corps augmentés
- T P5_21 - Environnements immersifs
- T P5_24 - Gouverner l'environnement...
- T P5_25 - Séminaire "Turing"
- T P5_26 - Choeur des normaliens
- T P5_27 - LabOrchestra
- T P5_28 - Atelier Régie
- T P5_29 - Atelier Théâtre "Se mettre e...

- T RT1 - ARIA
- T RT2 - ARTeQ
- T RT3 - ARRC
- T RT4 - ARéco

International

- S I1 - Deux mois à l'étranger I R
- S I2 - Semestre à l'étranger I A V
- T I3 - Seconde langue

Coloration

- T C1 - Administration
- T C2 - Expertise
- T C3 - Entrepreunariat et innovation
- T C4 - Engagement associatif ou électif
- T C5 - Connaissance du milieu indust... I

Pratique sportive

- T C6_1 - Basket
- T C6_2 - Badminton
- T C6_4 - Escalade
- T C6_5 - Football
- T C6_6 - Hand Ball
- T C6_7 - Rugby
- T C6_8 - Volley
- T C6_9 - Cheerleading
- T C6_10 - Course
- T C6_11 - Tennis
- T C6_3 - Danse

- T C7 - Pratique artistique
- S C0 - Approfondissement

ENS Paris-Saclay version 19 septembre 2022

Modularity in Interactive Theorem Provers

Arthur Adjedj (PhD Student)
Université Paris-Saclay, ENS Paris-Saclay

Nicolas Tabareau (Advisor)
Inria, France

Yannick Forster (Advisor)
Inria, France

Interactive theorem provers (ITPs) do not easily allow users to adapt and extend existing codebases without either changing the original code or duplicating it. The pattern of extending definitions occurs particularly often in programming language theory, type theory, and program verification, leading to a lot of code duplication and high maintainance burden. Existing approaches to this problem either require encoding datatypes as complicated expressions, obfuscating the development, or rely on meta-programming facilities which were underdeveloped in most ITPs until recently.
The goal of this research is to develop a principled way to produce modular code, leveraging the strengths of dependent types and modern meta-programming techniques.

Note. *This research topic has been collabaratively developed by Arthur Adjedj, Nicolas Tabareau and Yannick Forster. Arthur Adjedj will be co-supervised by Nicolas Tabareau and Yannick Forster and hosted in the Cambium team at the Centre Inria de Paris. Arthur Adjedj will conduct preliminary investigations on this topic, during a five-month internship under the supervision of Yannick Forster.*

Introduction

Interactive Theorem Provers (ITPs), also known as proof assistants, are tools which allow for the development and mechanical verification of formal proofs. ITPs can be used to provide a very high level of guarantees for software (in particular the complete absence of bugs), and several projects make use of proof assistants to verify real-world software, such as the CompCert C compiler (Leroy [2009]), the sel4 operating systems kernel (Lyons et al. [2025]), and AWS' authorization policy language, Cedar (AWS [2026]). They have also been used successfully to formalise landmark theorems in mathematics such as the Four Colour Theorem or the Feit Thompson theorem, and the use of proof assistants is on the rise in mathematics departments, with big mathematical developments like Lean's Mathlib library (The mathlib Community [2020]) getting more and more public attention.

Like other programming languages, ITPs fall victim to the the fact that reusing definitions can be non-trivial. This is generally referred to as the **expression problem** (Wadler [1998]):

"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."

In fact, reusing and adapting existing data structures and programs modularly is a challenge for which few solutions have been produced over the years in industrial programming languages (Oliveira and Cook [2012]). It is even worse for ITPs, where in addition to programs, one needs to adapt proofs and dependently typed programs as well. Most projects that try to extend existing formalisation resort to copying the original content and adapting it for its own purpose. Since proofs about programs are arguably even harder to maintain than programs, this copy-paste approach incurs a huge maintenance burden.

A central contribution on matters of modularity in regular programming languages, "Data Types à la Carte" (Swierstra [2008]), provides a simple and elegant solution to the expression problem based on a parametrical approach to modular syntax in Haskell. This solution, however, cannot be easily adapted to ITPs since ITPs needs to ensure consistency via their type system. Instead, existing approaches either rely on complex encodings of datatypes that leak to the user, or on meta-programming.

"Meta-Theory à la Carte" (Delaware et al. [2013]) and "Pyrosome" (Jamner et al. [2025]) base their modularity on internal encodings of types (namely, impredicative church-encodings in the former, and Generalized Algebraic Theories in the latter). In both cases, the constructions are inneficient, incapable of extending previously user-defined inductive types (*i.e.,* expecting users to rely on the aforementionned encodings from the ground up), and expose the underlying internals of the encodings to the user. Having to deal with encodings of types rather than types adds a heavy burden in particular for new users interested in program verification. The lesson to include inductive datatypes natively has been learned early by popular ITPs (namely Rocq and Lean), who at first had no primitive notions of inductive types in their systems and then resorted to add those. Nowadays, most systems usually posess a syntactic notion of (co-)inductive types, and justify the ability to define these via some classes of models, allowing users to work with the abstractions these types provide, rather than with their encoding in said models. The only popular system that still relies on encodings to define such types, and manages to hide their implementation details well, is Isabelle/HOL.

On the other hand, Rocq à la Carte (Forster and Stark [2020]) relies on the meta-programming capabilities offered by the MetaRocq Project (Sozeau et al. [2020]) to allow users to construct new inductive types and functions by merging other inductive types, and/or adding new constructors. New functions on a "merged" datatype can then be constructed by merging past functions. The metaprogram then uses the given piece of information to reconstruct a new inductive type, and new functions, based on the informations given by the user. While great for extending constructions "vertically" (*i.e.,* by adding constructors to a type), this system does not allow for horizontal extensions (*i.e.,* extending the type signature of inductive types and their constructors). Furthermore, this approach has been hindered in the past by the lack of good metaprogramming frameworks in ITPs.

None of these systems, independently of whether they use encodings or the meta-programming, handle all of the type-system of ITPs they are implemented for (*e.g.,* none handle co-inductive types), or allow users to extend past formalisations "after the fact". Instead, formalisations have to be built from the ground up with the expectation that they will be extended with a specific framework in mind, making them much less useful for real-world uses.

Objectives

The main objective for this PhD thesis will be to provide a better understanding of modularity in ITPs, and to produce useful tools to answer the expression problem along the way. This implies iterating on past attempts at solving the issue, developing principled ways to extend existing formalisations, and applying those to various case studies to refine the implementation. These tools will be implemented in the Lean 4 proof-assistant.

The end result of this research will be a generic tool for modularity that could take an arbitrary formalisation and allow the user to adapt it in various ways such as extending user-defined (co)-inducive types both vertically and horizontally, tweaking existing definitions, and providing the opportunity to correct proofs from the original formalisation.
To give an example of what modularity would look like in practice, let's look at the original example for the expression problem, written in Lean:

```
namespace Example
  structure EvalExp : Type where
    eval : Int

  def lit (n : Int) : EvalExp where
    eval := n

  def add (left right : EvalExp) : EvalExp where
    eval := left.eval + right.eval

  def ex : EvalExp := add (lit 1) (lit 2)
end Example
```

This code defines, in the namespace `Example`, a record type `EvalExp` containing a single field `eval : Int`, as well as two functions `lit` and `add` to construct `EvalExp`, the former constructing such an exp from a given `n : Int` and another taking two `EvalExp`s and returning the addition of their respective values. finally, we can compose those to construct a `ex : EvalExp`, evaluating `ex.eval` gives 3. We now want to modularly extend this code by adding two new fields `toString : String` and `toStringCorrect : toString.toIntEval = eval`. The former is a string representation of the value, and the second a proof that parsing and evaluating that string as an arithmetic expression returns the same value as `eval`.

```
modular Example as ExtendedExample --Extend the previous namespace by adding two fields to `EvalExp`
  extend EvalExp where
    toString : String
    toStringCorrect : toString.toIntEval = eval

  lit.toString (n : Int) := n.toString --Fill in the obligations
  lit.toStringCorrect (n : Int) := <some proof>

  add.toString (left right : EvalExp) := s!"{left.toString} + {right.toString}"
  add.toStringCorrect (left right : EvalExp) := <some proof>
```

Extending `EvalExp` with these fields necessitates populate the new fields for `lit` and `add`. Once that is done, the new, extended structure and definitions get added to the environment in the `ExtendedExample` namespace. In particular, `ExtendedExample.ex` gets automatically generated, with `ex.toString` having `"1 + 2"` as its value, and `ex.toStringCorrect` a proof that `"1 + 2".toIntEval = 3`.

In Lean, even though inductive types are a primitive notion of the system, co-inductive types are not. In particular, Lean provides stricly encapsulated co-inductive predicates encodings, *i.e.,* the encoded types behave just like primitive coinductive types would if they were implemented, and do not leak their implementation. We thus also want to provide good modular reasoning on such encodings. This includes looking into the denotational semantics of type systems, in particular that of inductive and co-inductive types, and studying modularity as a mathematical notion in relevant models. Examples include Categories with Families, extended with Theories of Signatures à la Kovács to encode complex inductive types (Kovács [2022]), and Quotient Polynomial Functors (Avigad et al. [2019]), for which there already exist an implementation in Lean 4

(Keizer [2026]). From a user-friendliness perspective, we may find inspiration for the syntax of our system in Marmaduke et al. [2025], which presents a type system with open types and open function, allowing one to easily extend inductive types vertically. Developing automations, in particular tactics, to allow users to fill in the holes of definitions and theorems easily after extending an inductive type is of particular importance, and may be related to works in the field of proof-repair (Gandhi et al. [2025]; Ringer [2021]).

The development of modular tooling will be made through experimentations on various case studies to both better understand the ways in which modularity can be used in practical settings, and to develop useful modular formalisations and libraries. Potential case studies can be split in 4 categories:

- Formalisation of programming languages and type theories: A given formalised type theory can then be extended by first extending its syntax and typing/reduction rules, and then adapting the logical relation to adapt to the added constructs. A good example, and potential case study, would be to formalise the Barendregt's Lambda Cube (Barendregt [1991]), starting with the Simply Typed Lambda-Calculus (STLC) as a basis, building the 3 base vertices of the cube on top of STLC, and merging these 3 in various ways to build the remaining vertices of the cube. Another would be to look at "Martin-Löf à la Coq" (Adjedj et al. [2024]), and the many extensions that were built on top of the base formalisation (Pédrot [2024], Baillon et al. [2026] and Laurent et al. [2024]). All of these formalisations are non trivial and have big codebases, containing about 30 000 lines of Rocq code each. Adapting the latter formalisations to "simply" be modular extensions of the former would be a great working example of real-world use.
- Compilers and interpreters: The modular construction and verification of interpreters has been studied in the past (Jamner et al. [2025], Michelland et al. [2024], Rest et al. [2022]), and serve as good basis for our case studies.
- Programming language semantics: Strata (Strata-org [2026]) is a recent in-development library that aims to offer a unified platform for formalizing language syntax and semantics. Users of the library can construct new intermediate representations, referred to as *dialects.* The correctness of optimization passes over a program written in a given dialect is then discharged to a SMT solver. Instead of extending the trusted code base and relying on (sometimes) slow SMT procedures, the system could be improved by providing static proofs of correctness for the optimization passes. Since one can construct a new dialect by extending another, allowing for modular proofs of correctness for the passes is desirable.
- Mathematics: While inductive types are used extensively in computer science, mathematics tend to rely more on record types, who come in many different flavours. Row types, and in particular row polymorphism (Hubers and Morris [2023]) allow for some modular reuse of functions and theorems over records types. As such, being able to replicate this behaviour in systems that do not support such polymorphism through metaprogramming encodings should be explored.

A tentative timeline for this PhD is as follows: Relying on a reimplementation of "Coq à la carte" in Lean 4, we will spend the first year focusing our efforts on providing a well-behaved notion and implementation of vertical and horizontal modularity, and use this to formalise Barendregt's Lambda-Cube, including the Calculus of Constructions. Once that is done, we will shift our focus towards being able to modularise existing non-modular developments such as Strata. Lastly, we will focus our attention on managing modular developments over types defined using encodings, such as Lean's co-inductive predicates.

Adequacy

We believe that Arthur Adjedj is well-suited for this topic because he has a background in dependently-typed proof assistants and type theory. He worked on "Martin-Löf à la Coq" (Adjedj et al. [2024]), a formalisation of Martin-Löf Type Theory in Rocq, during an internship under the supervision of Nicolas Tabareau and Loïc Pujet, as well as " AdapTT: Functoriality for Dependent Type Casts" (Adjedj et al. [2026]) under the supervision of Meven Lennon-Bertrand. The latter paper uncovers a general notion of type-casting in dependent type theory that exhibits the functoriality of type formers, encompassing a core construct which underlies many recent works in the field. He has furthermore contributed to the Lean 4 proof assistant for many years. All these works have provided him a deep understanding of both the theory and implementation of ITPs.

Nicolas Tabareau is a key member of the Rocq development team, and has worked on proof translation between Rocq and Lean. He has coordinated large formalisations of type theory in Rocq that would have benefited greatly from modular methods.

Yannick Forster has worked on meta-programming in Rocq for years (Sozeau et al. [2020], Liesnikov et al. [2020]) and both informally and formally supervised projects, including a past effort in providing modularity to Rocq (Forster and Stark [2020]). He has initiated the meta-programming rosetta stone project for Rocq. He is a member of the MetaRocq team and maintains strong connections to the development teams of Rocq (Matthieu Sozeau, Nicolas Tabareau, Hugo Herbelin), Rocq-Elpi (Enrico Tassi), Ltac2 (Pierre-Marie Pédrot, Gaëtan Gilbert), Agda (Jesper Cockx), and Lean (Sebastian Ullrich, Mario Carneiro)

References

Adjedj, A., Lennon-Bertrand, M., Benjamin, T. and Maillard, K. (2026) AdapTT: Functoriality for Dependent Type Casts, *Proc ACM Program Lang.* 2026;10(POPL)., doi:10.1145/3776664.

Adjedj, A., Lennon-Bertrand, M., Maillard, K., Pédrot, P.M. and Pujet, L. (2024) Martin-Löf à la Coq, In. *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2024, Association for Computing Machinery, London, UK 2024: pp. 230–245, doi:10.1145/3636501.3636951.

Avigad, J., Carneiro, M. and Hudon, S. (2019) Data Types as Quotients of Polynomial Functors, In: Harrison, J., O'Leary, J. and Tolmach, A. (eds.). *10th International Conference on Interactive Theorem Proving (ITP 2019)*. Vol 141. Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany 2019, doi:10.4230/LIPIcs.ITP.2019.6.

AWS (2026) Cedar Specification, 2026, https://github.com/cedar-policy/cedar-spec.

Baillon, M., Mahboubi, A. and Pédrot, P.M. (2026) In Cantor Space No One Can Hear You Stream, In. *ESOP2026 - 35th European Symposium on Programming and Systems*, Turin, Italy 2026, https://hal.science/hal-05495450.

Barendregt, H. (1991) An Introduction to Generalized Type Systems, *Journal of Functional Programming*. 1991;1: 125–154., doi:10.1017/S0956796800020025.

Delaware, B., S. Oliveira, B.C. d. and Schrijvers, T. (2013) Meta-theory à la carte, *SIGPLAN Not.* 2013;48(1): 207–218., doi:10.1145/2480359.2429094.

Forster, Y. and Stark, K. (2020) Coq à la carte: a practical approach to modular syntax with binders, In: Blanchette, J. and Hritcu, C. (eds.). *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, ACM 2020: pp. 186–200, doi:10.1145/3372885.3373817.

Gandhi, A., Tadipatri, A.R. and Gowers, T. (2025) Automatically Generalizing Proofs and Statements, In: Forster, Y. and Keller, C. (eds.). *16th International Conference on Interactive Theorem Proving (ITP 2025)*. Vol 352. Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany 2025: pp. 12:1–12:18, doi:10.4230/LIPIcs.ITP.2025.12.

Hubers, A. and Morris, J.G. (2023) Generic Programming with Extensible Data Types: Or, Making Ad Hoc Extensible Data Types Less Ad Hoc, *Proc ACM Program Lang.* 2023;7(ICFP)., doi:10.1145/3607843.

Jamner, D., Kammer, G., Nag, R. and Chlipala, A. (2025) Pyrosome: Verified Compilation for Modular Metatheory, *Proc ACM Program Lang.* 2025;9(OOPSLA2)., doi:10.1145/3763052.

Keizer, A. (2026) A (co)datatype package for Lean 4, based on Quotients of Polynomial Functors, 2026, https://github.com/alexkeizer/QpfTypes.

Kovács, A. (2022) *Type-Theoretic Signatures for Algebraictheories and Inductive Types*, phdthesis, 2022, doi:10.15476/ELTE.2022.070.

Laurent, T., Lennon-Bertrand, M. and Maillard, K. (2024) Definitional Functoriality for Dependent (Sub)Types, In: Weirich, S. (ed.). *33rd European Symposium on Programming, ESOP 2024*. Vol 14576. Lecture Notes in Computer Science, Springer 2024: pp. 302–331, doi:10.1007/978-3-031-57262-3_13.

Leroy, X. (2009) Formal verification of a realistic compiler, *Communications of the ACM*. 2009;52(7): 107–115., http://xavierleroy.org/publi/compcert-CACM.pdf.

Liesnikov, B., Ullrich, M. and Forster, Y. (2020) Generating induction principles and subterm relations for inductive types using MetaCoq, 2020, doi:10.48550/ARXIV.2006.15135.

Lyons, A., McLeod, K., Klein, G., et al. (2025) seL4/seL4: seL4 14.0.0, December 2025, doi:10.5281/zenodo.17904671.

Marmaduke, A., Ingle, A. and Morris, J.G. (2025) Understanding Haskell-style Overloading via Open Data and Open Functions, *arXiv e-prints*. advance online publication July 2025;arXiv:2507.16086., doi:10.48550/arXiv.2507.16086.

The mathlib Community (2020) The Lean Mathematical Library, In. *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2020, Association for Computing Machinery, New Orleans, LA, USA 2020: pp. 367–381, doi:10.1145/3372885.3373824.

Michelland, S., Zakowski, Y. and Gonnord, L. (2024) Abstract Interpreters: A Monadic Approach to Modular Verification, *Proc ACM Program Lang.* 2024;8(ICFP)., doi:10.1145/3674646.

Oliveira, B.C.d.S. and Cook, W.R. (2012) Extensibility for the Masses, In: Noble, J. (ed.). *ECOOP 2012 – Object-Oriented Programming*, Springer Berlin Heidelberg, Berlin, Heidelberg 2012: pp. 2–27.

Pédrot, P.M. (2024) "Upon This Quote I Will Build My Church Thesis", In. *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '24, Association for Computing Machinery, Tallinn, Estonia 2024, doi:10.1145/3661814.3662070.

Rest, C. van der, Poulsen, C.B., Rouvoet, A., Visser, E. and Mosses, P. (2022) Intrinsically-typed definitional interpreters à la carte, *Proc ACM Program Lang.* 2022;6(OOPSLA2)., doi:10.1145/3563355.

Ringer, T. (2021) *Proof Repair*, Doctoral dissertation, 2021.

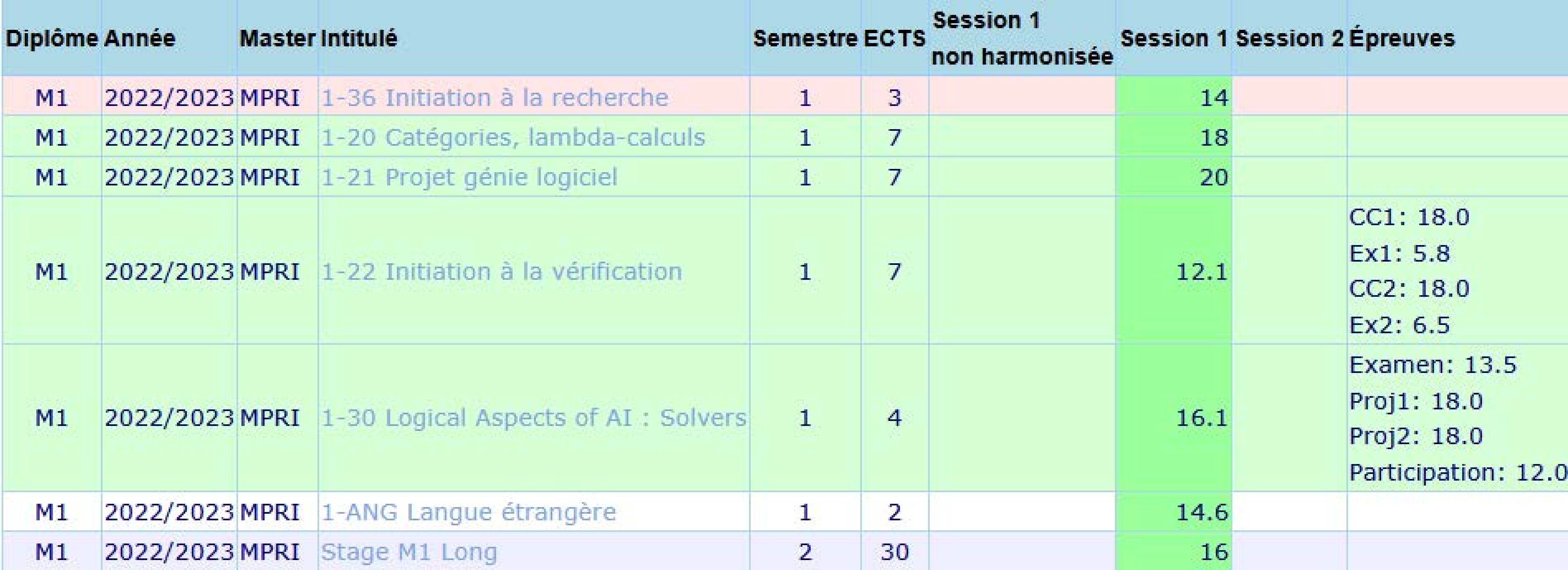
Sozeau, M., Anand, A., Boulier, S., et al. (2020) The MetaCoq Project, *Journal of Automated Reasoning*. advance online publication 2020., doi:10.1007/s10817-019-09540-0.

Strata-org (2026) Strata, 2026, https://github.com/strata-org/Strata.

Swierstra, W. (2008) Data types à la carte, *J Funct Program.* 2008;18(4): 423–436., doi:10.1017/S0956796808006758.

Wadler, P. (1998) The expression problem, November 1998, https://homepages.inf.ed.ac.uk/wadler/papers/expression/expression.txt.

Diplôme	Année	Master	Intitulé	Semestre	ECTS	Session 1 non harmonisée	Session 1	Session 2	Épreuves
L3	2021/2022	MPRI	L3-1-14 Algorithmique 1	1	6		13.6		EXSH: 16.0 CCSH: 18.7 EXJGL: 10.5 CCJGL: 10.7
L3	2021/2022	MPRI	L3-1-17 Calculabilité-Complexité	1	6		13		Partiel: 11.0 Examen: 15.0
L3	2021/2022	MPRI	L3-1-18 Programmation 1	1	6		14.7		DM-1: 14.5 CC: 15.5 Projet-1: 20.0 Projet-2: 17.0 Examen: 11.1 CC-2: 20.0
L3	2021/2022	MPRI	L3-1-12 Architecture et système	1	6		18.4		Projet 1: 19.3 Projet 2: 16.0 Examen: 20.0
L3	2021/2022	MPRI	L3-1-13 Mathématiques Discrètes	1	6		10.8		Partiel: 9.3 Examen: 8.5 Contrôle continu: 16.5
L3	2021/2022	MPRI	L3-2-01 Langages formels	2	6		11.3		CC1: 15.7 Ex1: 14.9 TP: 13.0 CC2: 8.3 Ex2: 4.7
L3	2021/2022	MPRI	L3-2-26 Logique	2	6		14.7		DM: 13.0 Exam: 15.5
L3	2021/2022	MPRI	L3-2-06 Projet Logique	2	6		17.1		
L3	2021/2022	MPRI	L3-2-21 Lambda calcul et logique informatique	2	6		12.7		DM: 14.5 Examen: 10.9
L3	2021/2022	MPRI	L3-2-22 Programmation 2	2	6		12.6		Projet Defonctionnalisation: 16.0 Projet BDD: 12.0 Exam: 11.1
L3	2021/2022	MPRI	L3-1-ANG Anglais	1	3		16.1		S1: 15.7 S2: 16.4
L3	2021/2022	MPRI	L3-1M-10 Algèbre 1	1	6		10		
L3	2021/2022	MPRI	L3-2-05 Projet Programmation 2	2	6		15		P1: 14.0 P2: 18.0 P3: 13.0
L3	2021/2022	MPRI	Stage de recherche L3	2	5		17		



Diplôme	Année	Master	Intitulé	Semestre	ECTS	Session 1 non harmonisée	Session 1	Session 2	Épreuves
M1	2022/2023	MPRI	1-36 Initiation à la recherche	1	3		14		
M1	2022/2023	MPRI	1-20 Catégories, lambda-calculs	1	7		18		
M1	2022/2023	MPRI	1-21 Projet génie logiciel	1	7		20		
M1	2022/2023	MPRI	1-22 Initiation à la vérification	1	7		12.1		CC1: 18.0 Ex1: 5.8 CC2: 18.0 Ex2: 6.5
M1	2022/2023	MPRI	1-30 Logical Aspects of AI : Solvers	1	4		16.1		Examen: 13.5 Proj1: 18.0 Proj2: 18.0 Participation: 12.0
M1	2022/2023	MPRI	1-ANG Langue étrangère	1	2		14.6		
M1	2022/2023	MPRI	Stage M1 Long	2	30		16		

Diplôme	Année	Master	Intitulé	Semestre	ECTS	Session 1 non harmonisée	Session 1	Session 2	Épreuves
M2	2025/2026	MPRI	AISAV Interprétation abstraite : application à l'analyses statique et à la vérification	1	6				Written exam: 14.0
M2	2025/2026	MPRI	ECOLO Exploration de modèles de calcul à l'aide de la logique linéaire	1	6				
M2	2025/2026	MPRI	FUN Programmation fonctionnelle et systèmes de types	1	6				Partiel: 17.5
M2	2025/2026	MPRI	HOTT Théorie des types homotopiques	1	3				
M2	2025/2026	MPRI	PRFA Assistants de preuves	1	3	17.8			Exam: 17.9 Project: 17.7
M2	2025/2026	MPRI	PRFSYS Fondements des systèmes de preuves	1	3				EXAMEN: 18.3
M2	2025/2026	MPRI	SEMPL Modèles des langages de programmation: domaines, catégories, jeux	1	6				Partiel: 15.0
M2	2025/2026	MPRI	SYNC Programmation synchrone de systèmes réactifs	1	3				
M2	2025/2026	MPRI	Stage de recherche M2	2	30				

Arthur Adjedj
4 Avenue d’Antibes
13008 Marseille
(33) 06 01 24 19 69
arthur.adjedj@ens-paris-saclay.fr

Paris, le 5 mars 2026

Madame, Monsieur,

Étudiant en informatique à l’ENS Paris-Saclay, et sur le point de compléter mon M2 MPRI, je souhaite poursuivre vers une thèse académique.

Il y a maintenant 5 ans, j'étais en classe préparatoire MPSI au Lycée Thiers à Marseille. J'avais jusqu'ici développé déjà un fort attrait pour les mathématiques et l'informatique, mais ne m'étais pas encore découvert de passion pour la recherche. C'est au cours de cette année-là que j'ai été exposé au monde des méthodes formelles, et en particulier celui des assistants de preuves et de la théorie des types. J'ai tout de suite été attiré par les promesses que les outils de vérification pouvaient offrir, et a alors déterminé tôt que je souhaiterais travailler dans ces domaines de recherche dans le futur. Cette décision m'a amenée à être admis à l'ENS Paris-Saclay en MP 3/2. Mon TIPE à l'époque était déjà un solveur et assistant de preuve basé sur le calcul de séquents LK.
Une fois admis, j'ai voué mon temps et mes études aux assistants de preuves et à la théorie des types, faisant ainsi des stages de recherches, respectivement auprès de chercheurs développant l'assistant de preuves Rocq à Nantes, où j'ai pu rencontrer Nicolas Tabareau et Yannick Forster, aujourd'hui directeurs potentiels pour ma thèse, puis auprès de Jesper Cockx, l'un des développeurs principaux d'Agda, avant de passer, l’année dernière, un an à Cambridge auprès de Meven Lennon-Bertrand, chercheur en théorie des types. J'ai dans mon temps libre aussi contribué des fixes et features à l'assistant Lean, et développé au cours du temps une expertise toute particulière pour ce système. Le résultat de ces travaux ont été deux papiers, acceptés respectivement à Certified Programs and Proofs (CPP) 2024, portant sur la formalisation et vérification de théories des types dans des théories des types, ainsi qu'à Principles of Programming Languages (POPL) 2026, portant sur la construction de théories des types exhibants les propriétés fonctorielles des constructeurs de type usuels.
La thèse proposée ici s'inscrit dans la continuité de mes travaux précédents. Cette formation saura m’aider à développer une plus grande expertise dans le domaine des assistants à la preuve et de la théorie des types, et me permettra d’élargir mes horizons pour ma recherche future.

Veuillez agréer, Madame, Monsieur, l'expression de mes salutations distinguées

To whom it may concern,

I am writing in recommendation of Arthur Adjedj for a PhD position. Arthur is currently a computer science student at ENS Paris-Saclay, following the renowned M2 MPRI program. His research interests center on type theory and implementation of proof assistants, which coincide with my own as one of the lead developers of the Lean theorem prover.

Arthur has shown not only interest but expertise in these topics through more than a dozen individual contributions to Lean. Many of them involved fixing or improving the generation of automatically-derived declarations, especially around corner cases in the internal, formal representation of Lean types, proving his in-depth understanding of both the theory and the implementation side. He did all of this self-sufficiently without any assistance from our side. In fact, I was positively shocked when I first learned he was not already a PhD student considering the complex and high-quality changes he had made.

While I did not have the chance so far to collaborate with Arthur more closely and so cannot comment on some of his other qualities, his technical craftsmanship leaves me with no doubt that he will be able to excel at any challenge given to him in this field and that he will have a role in shaping the future design and implementation of proof assistants. I recommend him without reservation.

Sincerely,

[illegible]

Dr.-Ing. Sebastian Ullrich
Head of Engineering & Co-Founder, Lean FRO
sebastian@lean-fro.org
Leopoldstr. 162, 80804 München, Germany

Arthur Adjedj

FORMATION

ENS Paris-Saclay— Diplôme de l'ENS, Master Parisien de Recherche en Informatique (MPRI)

September 2021 - August 2026

Research-oriented Computer Science degree. Focus on the relationships between logic and computations.

ISAE-SUPAERO — *Gap year*

September 2023 - August 2024

Master of Science in Aeronautical and Aerospace engineering.

Lycée Thiers, MPSI-MP*— *CPGE*

September 2019 - June 2021

Preparatory Classes for Grandes Écoles.

EXPERIENCE

University of Cambridge, Department of Computer Science and Technology — *Research placement*

September 2024 - August 2025

Study of the interactions between dependent subtyping and inductive families of types.

Airbus Helicopters— *Internship*

March 2024 - August 2024

Software Engineering

TU Delft Programming Languages Group— *M1 Research Internship*

February 2023 - July 2023

Implementation of an arithmetic solver for the unification of cumulative universe constraints in Agda.

SKILLS

Programming: Efficiency in Python, Java, C, C++,Scala, OCaml, Rust, Haskell, Lean, Agda, Coq

Interests in : Programming, Logic, Algorithms, Dependent Type Theory, Software Engineering, Verification, Programming Languages

Open Source contributions to the Lean 4 Programming Language and Theorem Prover

LANGUAGES

French: Mother tongue

English: Cambridge C2 Certification, TOEFL iBT 114/120pts

Spanish: B2 Level

Team Gallinette, Inria Rennes— *L3 Research Internship*

June 2022 - August 2022

Formalisation of a Type Theory inside a Type Theory, using the Coq proof-assistant.

Fleetenergies — *Data Science Internship*

October 2020 - January 2021

Extraction and analysis of fuel consumption-related data to study the efficiency of vehicle fleets.

PUBLICATIONS

AdapTT: Functoriality for Dependent Type Casts — Arthur Adjedj, Meven Lennon-Bertrand, Kenji Maillard, Thibaut Benjamin

2025

POPL 2026: Proceedings of the 53rd ACM SIGPLAN Symposium on Principles of Programming Languages

https://arxiv.org/abs/2507.13774

The ability to cast values between related types is a leitmotiv of many flavors of dependent type theory, such as observational type theories, subtyping, or cast calculi for gradual typing. These casts all exhibit a common structural behavior that boils down to the pervasive functoriality of type formers. We propose and extensively study a type theory, called AdapTT, which makes systematic and precise this idea of functorial type formers, with respect to an abstract notion of adapters relating types. Leveraging descriptions for functorial inductive types in AdapTT, we derive structural laws for type casts on general inductive type formers.

AdapTT: A Type Theory with Functorial Types— Arthur Adjedj, Meven Lennon-Bertrand, Kenji Maillard, Thibaut Benjamin

2025

Pre-Proceedings. Published for the Types 2025 Conference.
https://msp.cis.strath.ac.uk/types2025/TYPES2025-book-of-abstracts.pdf

Many type theoretic features, from subtyping to observational equality and cast calculi in gradual typing, center around the ability to cast values from one type to another. These casts all act in a conspicuously similar fashion, which actually corresponds to the fact that type formers in dependent type theory are functorial. We propose and extensively study a type theory, AdapTT, which makes systematic and precise this idea of functorial type formers

Martin-Löf à la Coq—Arthur Adjedj, Meven Lennon-Bertrand, Kenji Maillard, Pierre-Marie Pédrot, Loïc Pujet- **Distinguished Paper Award**

2024

CPP 2024: Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs.
arXiv:2310.06376

We present an extensive mechanization of the metatheory of Martin-Löf Type Theory (MLTT) in the Coq proof assistant.

Engineering logical relations for MLTT in Coq— Arthur Adjedj, Meven Lennon-Bertrand, Kenji Maillard, Loïc Pujet

2023

Pre-Proceedings. Published for the Types 2023 Conference. https://types2023.webs.upv.es/TYPES2023.pdf

We report on a mechanization in the Coq proof assistant of the decidability of conversion and type-checking for Martin-Löf Type Theory (MLTT), extending a previous Agda formalization.

PROJECTS

Proost — Proof assistant

2023

Production project, creation of a proof-assistant based on a novel type theory called the Observational Calculus of Constructions (CCobs+).
Grade : 20/20

Chessa.ml — Algorithmic study of groups

2022

Creation of a framework for efficient computations over finite groups. In particular. This system was tailored to study the generating sets of groups, with the intent of studying a yet-to-be-solved mathematical conjecture.

LK— SAT-Solver

2021

Implementation of a first-order-logic solver based on LK Sequent Calculus. Theoretical and practical study of the implementation, as well as its potential extensions.
Grade : 20/20