Rodolphe Lepigre | CURRICULUM VITÆ

(+33) 06 95 42 73 30 • rodolphe.lepigre@inria.fr • lepigre.fr

Born on 04/04/1990, postdoctoral researcher at Inria.

Research positions (since 2016)

Postdoctoral researcher at Inria (Deducteam project) **Sep. 2017 – Dec. 2018**
Université Paris - Saclay, ENS de Cachan, France
Laboratoire Spécification et Vérification, CNRS

Temporary research and teaching position (French ATER) **Sep. 2016 – Aug. 2017**
Université Savoie Mont Blanc, Chambéry, France
Laboratoire de mathématiques, CNRS, UFR SceM

PhD thesis [Lepigre2017PhD] (2013 – 2016)

Institution: Université Grenoble Alpes (financement sur bourse MESR)

Laboratory: Laboratoire de mathématiques, CNRS, Université Savoie Mont Blanc

Supervisors: Christophe Raffalli (main), Pierre Hyvernat, Karim Nour

Reviewers: Thierry Coquand (Göteborgs Universitet), Alexandre Miquel (UdelaR Uruguay)

Jury president: Laurent Regnier (Aix-Marseille Université)

Other jury members: Andreas Abel (Göteborgs Universitet), Frédéric Blanqui (Inria)

Semantics and Implementation of an Extension of ML for Proving Programs

In recent years, proof assistant have reached an impressive level of maturity. They have led to the certification of complex programs such as compilers and operating systems. Yet, using a proof assistant requires highly specialised skills and it remains very different from standard programming. To bridge this gap, we aim at designing an ML-style programming language with support for proofs of programs, combining in a single tool the flexibility of ML and the fine specification features of a proof assistant. In other words, the system should be suitable both for programming (in the strongly-typed, functional sense) and for gradually increasing the level of guarantees met by programs, on a by-need basis.

We thus define and study a call-by-value language whose type system extends higher-order logic with an equality type over untyped programs, a dependent function type, classical logic and subtyping. The combination of call-by-value evaluation, dependent functions and classical logic is known to raise consistency issues. To ensure the correctness of the system (logical consistency and runtime safety), we design a theoretical framework based on Krivine's classical realizability. The construction of the model relies on an essential property linking the different levels of interpretation of types in a novel way.

We finally demonstrate the expressive power of our system using our implementation, by proving properties of standard programs like the map function on lists or the insertion sort.

Education (2009 – 2013)

Internship under the supervision of Pierre Hyvernat **Chambéry, France**
A Classical Realizability Interpretation of Judgment Testing [Lepigre2013M2] *Spring 2013*
Five months, as part of the second year my masters

Université Joseph Fourrier / ENSIMAG **Grenoble, France**
M2R MoSIG – Parallel, Distributed and Embedded Systems, ranked 11 / 52 *2012 – 2013*
Second year in a Masters program in Computer Science

Internship under the supervision of Peter Dybjer **Göteborg, Sweden**
Testing Judgment of type theory [Lepigre2012M1] *Summer 2012*
Two months, as part of the first year of my Masters

Chalmers Tekniska Högskola (exchange student) **Göteborg, Sweden**
Computer Science – Algorithms, Languages and Logic, with 30 extra ECTS *2011 – 2012*
Masters program in Computer Science

Université Savoie Mont Blanc (year abroad) **Chambéry, France**
M1 STIC Informatique et Systèmes Coopératfs, ranked 1 / 24 *2011 – 2012*
First year in a Masters program in Computer Science

Université Savoie Mont Blanc **Chambéry, France**
Licence STIC Informatique, ranked 5 / 104 *2010 – 2011*
Bachelor in Computer Science

Université Savoie Mont Blanc **Chambéry, France**
DEUG de Mathématiques *2009 – 2010*
Diploma obtained after two year's study in mathematics

Teaching experience

Teaching assistant as a doctoral student **Chambéry, France**
Computer science department, Université Savoie Mont Blanc *2014 – 2016*
64 hours of teaching per year for two years

Temporary lecturer **Chambéry, France**
Computer science department, Université Savoie Mont Blanc *2016 – 2017*
192 hours (plus 32 overtime hours) of teaching as a French ATER

Selection of taught subjects

- Introduction to algorithms using Python
- Synchronization systems and processes
- C programming
- Operating systems
- Advanced algorithms
- Mathematics for computer science

Initiation to research in mathematics in a primary school

I took part in a "Maths à Modeler" project (`http://mathsamodeler.ujf-grenoble.fr`) with Marion Foare (PhD student) and Tom Hirschowitz (CNRS researcher) in 2015. We lead a class of fifth grade pupils (CM2) for two hours a week during eight weeks. Our aim was to introduce the pupils to research in mathematics through a tiling problem that was easy to state, but still relatively complex to attack (see `https://lepigre.fr/pavage.html`). At the end of the project the pupils presented their findings in our laboratory.

Research interests and contributions

My research revolves around the design and implementation of PML_2 [Lepigre2018], which is a programming language supporting program proving. The main idea is to extend an ML-like language, similar to OCaml or SML, with the means of specifying and proving equational properties of its own programs. We thus combine the flexibility of a full-fledged programming language with the great specification power of a proof assistant. Although the development of PML_2 is my leading goal, it is first and foremost an excuse for attacking difficult theoretical and practical questions. Of course, the answers to such questions go beyond the scope of PML_2, although I often present them in its light. In this sense, the implementation of the language can be seen as an experimentation platform for driving theoretical investigations. I strongly believe that mixing abstract theoretical questions with pragmatic implementation aspects mutually benefits to both fields. For example, the type system given in [Lepigre2016] could not be implemented in a natural way due to the fact that PML_2 is a so-called Curry-style language. As a result, we developed a new framework for dealing with such languages, including new techniques using subtyping, choice operators and circular proofs [LepRaf2018a]. In the following, I briefly highlight my most important contributions.

Classical realizability and observational equivalence

To prove the correctness of the PML_2 system (logical consistency and runtime safety), I designed a theoretical framework based on Krivine's classical realisability [Lepigre2016]. This semantical model accounts for the specificities of the system, and in particular the notion of program equivalence that is used for specifying computational behaviours. I thus defined a relation of observational equivalence over terms, which can be naturally expressed in the classical realizability settings. Indeed, quantifying over all the evaluation contexts simply amounts to quantifying over all the stacks in a Krivine abstract machine. Another specificity of the model lies in its call-by-value nature, which yields an interpretation with three layers, against two in the usual (call-by-name) presentation of Krivine's classical realizability. Indeed, a type A is interpreted by a set of (fully-evaluated) values $[\![A]\!]$, a set of stack (evaluation contexts), and a set of terms $[\![A]\!]^{\perp\perp}$ linked by an orthogonality relation. In some sense, this means that $[\![A]\!]^{\perp\perp}$ is the completion of $[\![A]\!]$ with terms computing elements of $[\![A]\!]$. It is essential for these sets to be closed under observational equivalence.

Dependent functions, effects and value restriction

In PML_2, program properties are specified and proved by manipulating terms as the first-order objects of the type system. In particular, this gives a way of expressing properties that should hold for all terms. For example, we can write $\forall n, \forall m, n + m \equiv m + n$ to express the commutativity of an addition function. However, the first-order quantification that is used here ranges over every possible values, without any type consideration. Whereas functions like addition are usually not defined on all values, but on a restricted domain like the natural numbers, and may crash when fed with something else. Hence, we often need to rely on a stronger, typed form of quantification, with which we can write $\forall n \in \mathbb{N}, \forall m \in \mathbb{N}, n + m \equiv m + n$. Such types correspond to a form of dependent function. For instance, the previous example is inhabited by functions taking two number x and y, and returning a proof of $x + y \equiv y + x$. In PML_2, working with such dependent functions types is delicate, because of the presence of effects. Indeed, as for polymorphism in ML, dependent functions require some restriction to preserve soundness. The usual *value restriction* used in ML also applies here, but it is not satisfactory. Indeed, it restricts the application of dependent function to values, which breaks

the compositionality of the system. To solve this issue, I proposed to relax value restriction using the idea that a term that is (observationally) equivalent to a value can be considered to be a value [Lepigre2016]. Although this so-called *semantical value restriction* is simple, it is extremely hard to justify in the model. Indeed, it requires an essential property relating the different levels of interpretation of types in a novel way. As mentioned previously, a type A is interpreted both as a set of values $[\![A]\!]$ and as a set of terms $[\![A]\!]^{\perp\perp}$ defined as a completion of $[\![A]\!]$, which implies $[\![A]\!] \subseteq [\![A]\!]^{\perp\perp}$. The property that is required for justifying the semantical value restriction is the following: every value of $[\![A]\!]^{\perp\perp}$ should already be in $[\![A]\!]$. In other words, the completion operation on sets of values should be closed for values. To obtain this property, I extended the programming language with a new instruction, which provides new tests for observational equivalence [Lepigre2016; Lepigre2017PhD].

Type-checking and subtyping in Curry-style languages

Type checking (verifying that a given term inhabits a given type) and type inference (finding a type that is inhabited by a given term) tend to be undecidable in Curry-style languages like System F or PML_2. As a consequence, these system are sometimes considered impractical, although practicality and decidability are two different problems. The main issue with Curry-style languages is that their type systems are generally not syntax-directed, meaning that they cannot be easily implemented. In particular, there is no canonical way of deciding what typing rule should be applied first when attempting to prove a typing judgment. To solve this problem, we designed (with Christophe Raffalli) a framework based on subtyping [LepRaf2018a]. The main, innovating idea is to use a ternary relation $t \in A \subset B$ instead of the usual binary relation $A \subset B$. We interpret the former as the implication "if the term t has type A, then it also has type B", while the latter is interpreted as the inclusion "every element of type A is an element of type B". In the obtained system, only one typing rule applies for every term constructor, and only one subtyping rule applies for every pair of types (up to commutation). In particular, the connectives that do not have algorithmic contents (those that are not reflected in the syntax of the terms) are handled using subtyping exclusively. Such connectives include the quantifiers, but also the equality types of PML_2 and the least and greatest fixpoint constructors used by inductive and coinductive types.

Choice operators for a closed semantics

Our work on Curry-style languages [LepRaf2018a] not only involves a new notion of subtyping, but also a new way of dealing with variables based on choice operators, inspired by Hilbert's Epsilon operator. With this new presentation, bound variables are systematically substituted by closed symbols playing the role of witnesses for semantical properties. As a consequence, terms and types remain closed throughout the typing and subtyping rules, and the usual typing contexts are not requires anymore. This presentation has several advantages, the first of which being a simplification of the semantics. Indeed, free variables are usually handled (in realizability models) using so-called valuations, assigning them a semantic value. In our presentation, such maps are not required since all the necessary information is carried by the symbolic witnesses that are substituted to free variables. As an indirect consequence of this technique, structural rules such as weakening become completely transparent, and implementation only requires first-order unification. Finally, the elimination of contexts facilitates the construction of circular proofs (see below).

Circular proofs and termination checking

As mentioned earlier, the framework based on subtyping that we defined (with Christophe Raffalli) is compatible with inductive and coinductive types. They are added to the syntax of types in the form of least and greatest fixpoint operators, which are annotated by ordinals to form sized-types. To handle these constructors in a sound way, we introduce a notion of circular proof with a related well-foundedness criterion. In particular, the subtyping rules that are given for the fixpoints induce a form of infinite unfolding. This is due to the fact that we work with symbolic ordinals that are not given a concrete value, and it is thus impossible to know when the zero ordinal will be reached in a decreasing sequence. However, assuming that the proof is well-founded, we know that it will indeed be reached after finitely many unfolding steps. To keep the proof representation finite, we introduce circularity in our proofs, and check that they are well-formed using the size-change principle of Lee, Jones and Ben Amram. The notion of circular proof that we consider is in fact very general, and can also be applied to typing rules in our context. This allows us to type recursive programs in a very simple way, while proving their termination. The normalisation proof is obtained using standard reducibility candidates (or realizability) techniques. Indeed, as the structure of our circular proofs is well-founded, it is still possible to reason by (well-founded) induction on the structure of our typing (or subtyping) derivations.

Implementation tools and prototype languages

My work on the design of type systems for programming languages and proof assistants has resulted in a relatively large amount of implementation work. The most important projects are listed in the following section, and include PML_2 (which is described in [Lepigre2018] and [Lepigre2017PhD]), SubML (which is described in [LepRaf2018a]) but also a more recent work on a new implementation of Dedukti (Dowek et al.) which is still in progress. The implementation of such systems has also directed me toward the development of specific tools and software libraries providing convenient abstractions. For instance, I take part in the development of the Bindlib library [LepRaf2018b], that was initiated by Christophe Raffalli, and that provides an abstract representation for binders (which are very common in languages and proofs systems). We also developed (with Christophe Raffalli) a system for writing parsers called Earley, that is integrated to OCaml using a BNF-like syntax extension.

Software projects

I am involved in the development of several software projects. Most of them are related to my research activities, or ease the development of programming languages and proof assistants.

The SubML language (~6000 lines of code)

The SubML language implements the type system presented in [LepRaf2018a]. Its many features include subtyping, inductive and coinductive types, polymorphism, existential types, sized types and a termination checker. This is joint work with Christophe Raffalli.

`https://github.com/rlepigre/subml` (source code)
`https://rlepigre.github.io/subml` (online interpreter)

The PML_2 language (~8000 lines of code)

The PML_2 language implements the system presented in [Lepigre2017PhD]. It can be seen as an extension of SubML with proof of programs, classical logic and higher-order. Examples of programs and proofs of programs are given in [Lepigre2017PhD; Lepigre2018]. I was the only

author of PML_2 (up to minor fixes and examples) until the PhD thesis version (see below). Christophe Raffalli contributes to the development of the language since September 2017.

`https://github.com/rlepigre/pml/archive/thesis.tar.gz` (thesis version)
`https://github.com/rlepigre/pml` (latest source code)

The Lambdapi language (~1700 lines of code)

Lambdapi is an implementation of the $\lambda\Pi$-calculus modulo rewriting similar to Dedukti (Dowek et al.), but based on the Bindlib library (see below). It is intended to become the new implementation of Dedukti in the near future. The code is shorter by half, well documented, and first experiments indicate a small gain in terms of performances.

`https://github.com/rlepigre/lambdapi` (source code)

The OCaml Bindlib library for bound variables (~1300 lines of code)

Bindlib is an OCaml library providing tools for the manipulation of data structures with bound variables (e.g., λ-calculus, quantified formulas). It allows for efficient substitution and supports variable renaming. The project was initiated by Christophe Raffalli. I contributed several simplifications, improvements, new features, and most of the current documentation.

`https://github.com/rlepigre/ocaml-bindlib` (source code)

Earley parser combinator library for OCaml (~10000 lines of code)

Earley is a parser combinator library for the OCaml language. It relies on Earley's parsing algorithm and it is intended to be used in conjunction with a preprocessor, which provides an OCaml syntax extension for allowing the definition of parsers inside the OCaml source code. This is joint work with Christophe Raffalli, following [LepRaf2015] in which a different parsing technology was used.

`https://github.com/rlepigre/ocaml-earley`

Timed references for imperative state in OCaml (~500 lines of code)

The Timed library allows the encapsulation of reference updates in an abstract notion of state. It can be used to emulate a pure interface while working with (imperative) references. This is joint work with Christophe Raffalli.

`https://github.com/rlepigre/ocaml-timed`

Imagelib library for OCaml (~2300 lines of code)

The Imagelib library implements image formats without relying on any C library, which makes it suitable for compilation using `js_of_ocaml`. Supported image formats are PNG (RCF 2083), PPM, PGM, and PBM. The formats JPG, GIF and XCF are only partially supported.

`https://github.com/rlepigre/ocaml-imagelib` (source code)

Patoline, a modern typesetting system in OCaml (~60000 lines of code)

Patoline aims at providing an alternative to TEX-based systems by relying on a high-level programming language: OCaml. The project was initiated by Pierre-Étienne Meunier, Christophe Raffalli and Tom Hirschowitz. I joined the project in 2013 and became one of its main contributors. I notably used Patoline to produce my thesis [Lepigre2017PhD].

`https://github.com/patoline/patoline`

Invited talks, paper presentations and seminars

Invited speaker at the "Journées inaugurales du GT Scalp" — **Paris, France**
Une vue d'ensemble de PML_2 : realisabilité, sous-typage et preuves cycliques — *27/11/2018*

Gallium seminar (Inria Paris) — **Paris, France**
Abstract Representation of Binders using the Bindlib Library — *01/10/2018*

Paper presentation at the LFMTP Workshop — **Oxford, UK**
Representation of Binders Using the Bindlib (OCaml) Library — *07/07/2018*

Invited speaker at the Realizability Workshop (CIRM) — **Marseille, France**
The PML Language: Realizability at the Service of Program Proofs — *13/06/2018*

Gallium seminar (Inria Paris) — **Paris, France**
The PML_2 Language: Proving Programs in ML — *08/03/2018*

Type theory and realizability seminar (IRIF) — **Paris, France**
Practical Curry-Style using Choice Operators and Local Subtyping — *17/01/2018*

Seminar of the LCR team (LIPN) — **Villetaneuse, France**
Circular Proofs for Subtyping and Termination — *22/09/2017*

Contributed talk at the TYPES conference — **Budapest, Hungary**
Theory and Demo of PML2: Proving Programs in ML — *01/06/2017*

Visit to the Parsifal Team (Inria Saclay) — **Saclay, France**
PML2, Semantical Value Restriction & Pointed Subtyping — *22/02/2017*

Séminaire Logique et Interactions (I2M) — **Marseille, France**
Proofs of programs and subtyping in PML2 — *16/02/2017*

Paper presentation at the ESOP conference — **Eindhoven, Netherlands**
A Classical Realizability Model for a Semantical Value Restriction — *07/04/2016*

Logic and Semantics Seminar (Computer Laboratory) — **Cambridge, UK**
A call-by-value realizability model for PML — *26/02/2016*

Séminaire CHoCoLa (ENS de Lyon) — **Lyon, France**
A call-by-value realizability model for PML — *03/12/2015*

Short paper presentation at the JFLA workshop — **Val d'Ajol, France**
Mêler combinateurs et BNF pour l'analyse syntaxique en OCaml — *08/01/2015*

Visit to the Equipo de Lógica, Facultad de Ingeniería, Udelar — **Montevideo, Uruguay**
Toward an Adequation Lemma for PML2 — *03/12/2014*

Talk at the GaLoP workshop — **Grenoble, France**
Realizability, Testing and Game Semantics — *13/04/2014*

Organization of scientific events and community service

Workshop on *Termination and Circular Proofs* (main organizer) — **Chambéry, France**
`https://lama.univ-smb.fr/~lepigre/termination/` — *19/07/2017*

Workshop on *Coinduction in Type Theory* (co-organizer) — **Chambéry, France**
`https://lama.univ-savoie.fr/~hirschowitz/CoTT2017` — *03/07/2017*

***Parametricity, Logical Relations & Realizability* (CSL affiliated workshop)** — **Marseille, France**
`https://www.lama.univ-savoie.fr/pagesmembres/hyvernat/PLRR2016/` — *02/09/2016*

Géocalisation à Chambéry **(co-organizer, GeoCal working group meeting)** **Chambéry, France**
https://lama.univ-savoie.fr/~hirschowitz/Geocalisation2015 *08/06/2015*

Journées LAC à Chambéry **(co-organizer, LAC working group meeting)** **Chambéry, France**
https://lama.univ-savoie.fr/~hirschowitz/LAC2014 *20/11/2014*

Reviews for international journals and conferences

I wrote reviews for the TOPLAS international journal of the ACM in 2017 and 2018. I also wrote reviews for international conferences (POPL 2017, FoSSaCS 2018, etc.).

Supervision of interns

Building rewriting proofs in Dedukti **(Masters student)** **Cachan, France**
Supervision of Aristomenis-Dionysios Papadopoulo (with F. Blanqui) *5 months in 2018*

Developing and interface to Dedukti **(Engineering student)** **Cachan, France**
Supervision of Ismail Lachheb (with F. Blanqui and E.-J. Arias Gallego) *4 months in 2018*

A compiler for the MiniPML language **(Masters student)** **Chambéry, France**
Supervision of Christopher Dubois (with C. Raffalli) *4 months in 2015*

Publications

[CLS2019] Simon Colin, Rodolphe Lepigre, and Gabriel Scherer. "Unboxing Mutually Recursive Type Definitions". In: *Submitted to JFLA*. 2019. URL: https://lepigre.fr/files/publications/CSL2019.pdf.

[Lepigre2012M1] Rodolphe Lepigre. "Testing judgments of type theory". MA thesis. Chalmers Tekniska Högskola, Göteborg, Sweden, 2012. URL: https://lepigre.fr/files/publications/Lepigre2012M1.pdf.

[Lepigre2013M2] Rodolphe Lepigre. "A Classical Realizability Interpretation of Judgement Testing". MA thesis. Grenoble INP and Université Joseph Fourrier, Grenoble, France, 2013. URL: https://lepigre.fr/files/publications/Lepigre2013M2.pdf.

[Lepigre2016] Rodolphe Lepigre. "A Classical Realizability Model for a Semantical Value Restriction". In: *Proceedings of ESOP 2016, Eindhoven, The Netherlands*. Ed. by Peter Thiemann. 2016, pp. 476–502. URL: https://doi.org/10.1007/978-3-662-49498-1_19.

[Lepigre2017PhD] Rodolphe Lepigre. "Semantics and Implementation of an Extension of ML for Proving Programs". PhD thesis. Grenoble Alpes University, France, 2017. URL: https://tel.archives-ouvertes.fr/tel-01590363.

[Lepigre2018] Rodolphe Lepigre. "PML_2: Integrated Program Verification in ML". In: *Post-proceedings of TYPES, Budapest, Hungary*. LIPIcs. 2018. URL: https://lepigre.fr/files/publications/Lepigre2018.pdf.

[LepRaf2015] Rodolphe Lepigre and Christophe Raffalli. "Mêler combinateurs, continuations et EBNF pour une analyse syntaxique efficace en OCaml". In: *Journées Francophones dans Langages Applicatifs (short paper only)*. 2015. URL: https://lepigre.fr/files/publications/LepRaf2015.pdf.

[LepRaf2018a] Rodolphe Lepigre and Christophe Raffalli. “Practical Subtyping for Curry-Style Languages”. In: *ACM Trans. Program. Lang. Syst., TOPLAS* (2018). URL: `https://lepigre.fr/files/publications/LepRaf2018a.pdf`.

[LepRaf2018b] Rodolphe Lepigre and Christophe Raffalli. “Abstract Representation of Binders in OCaml using the Bindlib Library”. In: *Proceedings of LFMTP 2018, Oxford, UK*. Ed. by Frédéric Blanqui and Giselle Reis. Vol. 274. EPTCS. 2018, pp. 42–56. URL: `https://doi.org/10.4204/EPTCS.274.4`.