INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team π r 2

Design, study and implementation of languages for proofs and programs

Paris - Rocquencourt

Theme : Programs, Verification and Proofs

Activity Report

2010

Table of contents

πr^2 is a common project with University Paris 7, within the laboratory "Preuves, Programmes et Systèmes", which is itself joint between Paris 7 and CNRS.. The team has been created on January the 1st, 2009 and became an INRIA "équipe-projet" on July the 1st, 2009.

1. Team

Research Scientists

Pierre-Louis Curien [Team leader, DR CNRS, HdR]
Hugo Herbelin [DR INRIA, HdR]
Alexis Saurin [CR CNRS]
Matthieu Sozeau [CR INRIA]

Faculty Members

Pierre Letouzey [MC Paris 7]
Yann Régis-Gianas [MC Paris 7]
Bruno Bernardo [ATER Paris 7]

Technical Staff

Vincent Gross [Ingénieur ADT Coq]

PhD Students

Pierre Boutillier [Contrat doctoral université Paris 7]
Stéphane Glondu [Allocation couplée, inscrit à Paris 7]
Danko Ilik [Allocation Gaspard Monge, Ecole Polytechnique]
Guillaume Munch-Maccagnoni [Contrat doctoral université Paris 7]
Matthias Puech [Cotutelle avec l'Université de Bologne, financement Ministère de la Recherche italien (MIUR)]
Ronan Saillard [Grant from the european project "CerCo"]
Vincent Siles [Allocation couplée, inscrit à l'Ecole Polytechnique]
Élie Soubiran [Financement Ile de France, inscrit à l'Ecole Polytechnique]

Post-Doctoral Fellows

Nicolas Ayache [Postdoc of the european project "CerCo"]
Jeffrey Sarnat [INRIA funding]
Noam Zeilberger [Postdoc of the Foundation "Sciences Mathématiques de Paris"]

Visiting Scientists

Andreas Abel [Assistant Professor at LMU, Munich, visiting from October 1st, 2009 till March 31, 2010, funded by INRIA]
Martin Hofmann [Professor at LMU, Munich, visiting from November 14 to December 14, funded by INRIA]

Administrative Assistant

Cécile Espiègle

2. Overall Objectives

2.1. Overall Objectives

πr^2 is a two-years old joint INRIA team, which is devoted both to the study of foundational aspects of formal proofs and programs and to the development of the Coq proof assistant software, with a focus on the dependently typed programming language aspects of Coq. The team is part of the laboratory Proofs, Programs and Systems lab (PPS - UMR 7126, CNRS and Paris 7). Its explicit association with both University Paris 7 and CNRS is under way. The team acts as one of the strongest teams involved in the development of Coq as it hosts in particular the current coordinator of the Coq development team.

3. Scientific Foundations

3.1. Proof theory and the Curry-Howard correspondence

3.1.1. Proofs as programs

Proof theory is the branch of logic devoted to the study of the structure of proofs. An essential contributor to this field is Gentzen [45] who developed in 1935 two logical formalisms that are now central to the study of proofs. These are the so-called "natural deduction", a syntax that is particularly well-suited to simulate the intuitive notion of reasoning, and the so-called "sequent calculus", a syntax with deep geometric properties that is particularly well-suited for proof automation.

Proof theory gained a remarkable importance in computer science when it became clear, after genuine observations first by Curry in 1958 [42], then by Howard and de Bruijn at the end of the 60's [50], [65], that proofs had the very same structure as programs: for instance, natural deduction proofs can be identified as typed programs of the ideal programming language known as λ-calculus.

This proofs-as-programs correspondence has been the starting point to a large spectrum of researches and results contributing to deeply connect logic and computer science. In particular, it is from this line of work that Coquand's Calculus of Constructions [39] stemmed out – a formalism that is both a logic and a programming language and that is at the source of the Coq system [38].

3.1.2. Towards the calculus of constructions

The λ-calculus, defined by Church [37], is a remarkably succinct model of computation that is defined via only three constructions (abstraction of a program with respect to one of its parameters, reference to such a parameter, application of a program to an argument) and one reduction rule (substitution of the formal parameter of a program by its effective argument). The λ-calculus, which is Turing-complete, i.e. which has the same expressiveness as a Turing machine (there is for instance an encoding of numbers as functions in λ-calculus), comes with two possible semantics referred to as call-by-name and call-by-value evaluations. Of these two semantics, the first one, which is the simplest to characterise, has been deeply studied in the last decades [33].

For explaining the Curry-Howard correspondence, it is important to distinguish between intuitionistic and classical logic: following Brouwer at the beginning of the 20th century, classical logic is a logic that accepts the use of reasoning by contradiction while intuitionistic logic proscribes it. Then, Howard's observation is that the proofs of the intuitionistic natural deduction formalism exactly coincide with programs in the (simply typed) λ-calculus.

A major achievement has been accomplished by Martin-Löf who designed in 1971 a formalism, referred to as modern type theory, that was both a logical system and a (typed) programming language [56].

In 1985, Coquand and Huet [39], [40] in the Formel team of INRIA-Rocquencourt explored an alternative approach based on Girard-Reynolds' system F [46], [62]. This formalism, called the Calculus of Constructions, served as logical foundation of the first implementation of Coq in 1984. Coq was called CoC at this time.

3.1.3. The Calculus of Inductive Constructions

The first public release of CoC dates back to 1989. The same project-team developed the programming language Caml (nowadays coordinated by the Gallium team) that provided the expressive and powerful concept of algebraic data types (a paragon of it being the type of list). In CoC, it was possible to simulate algebraic data types, but only through a not-so-natural not-so-convenient encoding.

In 1989, Coquand and Paulin [41] designed an extension of the Calculus of Constructions with a generalisation of algebraic types called inductive types, leading to the Calculus of Inductive Constructions (CIC) that started to serve as a new foundation for the Coq system. This new system, which got its current definitive name Coq, was released in 1991.

In practice, the Calculus of Inductive Constructions derives its strength from being both a logic powerful enough to formalise all common mathematics (as set theory is) and an expressive richly-typed functional programming language (like ML but with a richer type system, no effects and no non-terminating functions).

3.2. The development of Coq

Since 1984, about 40 persons have contributed to the development of Coq, out of which 7 persons have contributed to bring the system to the place it is now. First Thierry Coquand through his foundational theoretical ideas, then Gérard Huet who developed the first prototypes with Thierry Coquand and who headed the Coq group until 1998, then Christine Paulin who was the main actor of the system based on the CIC and who headed the development group from 1998 to 2006. On the programming side, important steps were made by Chet Murthy who raised Coq from the prototypical state to a reasonably scalable system, Jean-Christophe Filliâtre who turned to concrete the concept of a small trustful certification kernel on which an arbitrary large system can be set up, Bruno Barras and Hugo Herbelin who, among other extensions, reorganised Coq on a new smoother and more uniform basis able to support a new round of extensions for the next decade.

The development started from the Formel team at Rocquencourt but, after Christine Paulin got a position in Lyon, it spread to École Normale Supérieure de Lyon. Then, the task force there globally moved to the University of Orsay when Christine Paulin got a new position there. On the Rocquencourt side, the part of Formel involved in ML moved to the Cristal team (now Gallium) and Formel got renamed into Coq. Gérard Huet left the team and Christine Paulin started to head a Coq team bilocalised at Rocquencourt and Orsay. Gilles Dowek became the head of the team which was renamed into LogiCal. Following Gilles Dowek who got a position at École Polytechnique, LogiCal globally moved to Futurs with a bilocalisation on Orsay and Palaiseau. It then split again giving birth to ProVal. At the same time, the Marelle team (formerly Lemme, formerly Croap) which has been a long partner of the Formel team, invested more and more energy in both the formalisation of mathematics in Coq and in user interfaces for Coq.

After various other spreadings resulting from where the wind pushed former PhD students, the development of Coq got definitely multi-site with the development now realised by employees of INRIA, the CNAM and Paris 7.

We next briefly describe the main components of Coq.

3.2.1. The underlying logic and the verification kernel

The architecture adopts the so-called de Bruijn principle: the relatively small *kernel* of Coq ensures the correctness of the proofs validated by the system. The kernel is rather stable with modifications tied to the evolution of the underlying Calculus of Inductive Constructions formalism. The kernel includes an interpreter of the programs expressible in the CIC and this interpreter exists in two flavours: a customisable lazy evaluation machine written in Objective Caml and a call-by-value bytecode interpreter written in C dedicated to efficient computations. The kernel also provides a module system.

3.2.2. Programming and specification languages

The concrete user language of Coq, called *Gallina*, is a high-level language built on top of the CIC. It includes a type inference algorithm, definitions by complex pattern-matching, implicit arguments, mathematical notations and various other high-level language features. This high-level language serves both for the development of programs and for the formalisation of mathematical theories. Coq also provides a large set of commands. Gallina and the commands together forms the *Vernacular* language of Coq.

3.2.3. Libraries

Libraries are written in the vernacular language of Coq. There are libraries for various arithmetical structures and various implementations of numbers (Peano numbers, implementation of $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ with binary digits, implementation of $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ using machine words, axiomatisation of $\mathbb{R}$). There are libraries for lists, list of a specified length, sorts, and for various implementations of finite maps and finite sets. There are libraries on relations, sets, orders.

3.2.4. Tactics

The tactics are the methods available to conduct proofs. This includes the basic inference rules of the CIC, various advanced higher level inference rules and all the automation tactics. Regarding automation, there are tactics for solving systems of equations, for simplifying ring or field expressions, for arbitrary proof search, for semi-decidability of first-order logic and so on. There is also a powerful and popular untyped scripting language for combining tactics into more complex tactics.

Note that all tactics of Coq produce proof certificates that are checked by the kernel of Coq. As a consequence, possible bugs in proof methods do not hinder the confidence in the correctness of the Coq checker. Note also that the CIC being a programming language, tactics can be written (and certified) in the own language of Coq if needed.

3.2.5. Extraction

Extraction is a component of Coq that maps programs (or even computational proofs) of the CIC to functional programs (in Objective Caml, Scheme or Haskell). Especially, a program certified by Coq can further be extracted to a program of a full-fledged programming language then benefiting of the efficient compilation, linking tools, profiling tools, ... of the target software.

3.3. Dependently typed programming languages

3.3.1. The emergence of dependently typed programming

Dependently typed programming (shortly DTP) is an emerging concept referring to the diffuse and broadening tendency to develop programming languages with type systems able to express program properties finer than the usual information of simply belonging to specific data-types. The type systems of dependently-typed programming languages allow to express properties *dependent* of the input and the output of the program (for instance that a sorting program returns a list of same size as its argument). Typical examples of such languages were the Cayenne language, developed in the late 90's at Chalmers University in Sweden and the DML language developed at Boston. Since then, various new tools have been proposed, either as typed programming languages whose types embed equalities (Ωmega at Portland, ATS at Boston, ...) or as hybrid logic/programming frameworks (Agda at Chalmers University, Twelf at Carnegie, Delphin at Yale, OpTT at U. Iowa, Epigram at Nottingham, ...).

DTP contributes to a general movement leading to the fusion between logic and programming. Coq, whose language is both a logic and a programming language which moreover can be extracted to pure ML code plays a role in this movement and some frameworks for DTP have been proposed on top of Coq (Concoqtion at Rice and Colorado, Ynot at Harvard, Why in the ProVal team at INRIA). It also connects to Hoare logic, providing frameworks where pre- and post-conditions of programs are tied with the programs.

3.3.2. Issues regarding dependently typed programming

DTP approached from the programming language side generally benefits of a full-fledged language (e.g. supporting effects) with efficient compilation. DTP approached from the logic side generally benefits of an expressive specification logic and of proof methods so as to certify the specifications. The weakness of the approach from logic however is generally the weak support for effects or partial functions.

3.3.2.1. Type-checking and proof automation

In between the decidable type systems of conventional data-types based programming languages and the full expressiveness of logically undecidable formulae an active field of research explores a spectrum of decidable or semi-decidable type systems for possible use in dependently programming languages. At the beginning of the spectrum, this includes for instance the system F's extension ML_F of the ML type system or the generalisation of abstract data types with type constraints (G.A.D.T.) such as found in the Haskell programming language. At the other side of the spectrum, one finds arbitrary complex type specification languages (e.g. that a sorting function returns a list of type "sorted list") for which more or less powerful proof automation tools (generally first-order ones) exist.

3.3.2.2. Libraries

Developing libraries for programming languages takes time and generally benefits of a critical mass effect. An advantage is given to languages that start from well-established existing frameworks for which a large panel of libraries exist. Coq is such a framework.

3.4. Around and beyond the Curry-Howard correspondence

For two decades, the Curry-Howard correspondence was limited to the intuitionistic case but in 1990, an important stimulus spurred on the community following the discovery by Griffin that the correspondence was extensible to classical logic. The community then started to investigate unexplored potential fields of connection between computer science and logic. One of these fields was the computational understanding of Gentzen's sequent calculus while another one was the computational content of the axiom of choice.

3.4.1. Control operators and classical logic

Indeed, a significant extension of the Curry-Howard correspondence has been obtained at the beginning of the 90's thanks to the seminal observation by Griffin [47] that some operators known as control operators were typable by the principle of double negation elimination ($\neg\neg A \Rightarrow A$), a principle which provides classical logic.

Control operators are operators used to jump from one place of a program to another place. They were first considered in the 60's by Landin [54] and Reynolds [61] and started to be studied in an abstract way in the 80's by Felleisen *et al* [43], culminating in Parigot's $\lambda\mu$-calculus [59], a reference calculus that is in fine Curry-Howard correspondence with classical natural deduction. In this respect, control operators are fundamental pieces of the full connection between proofs and programs.

3.4.2. Sequent calculus

The Curry-Howard interpretation of sequent calculus started to be investigated at the beginning of the 90's. The main technicality of sequent calculus is the presence of *left introduction* inference rules and two kinds of interpretations of these rules are applicable. The first approach interprets left introduction rules as construction rules for a language of patterns but it does not really address the problem of the interpretation of the implication connective. The second approach, started in 1994, interprets left introduction rules as evaluation context formation rule. This line of work culminated in 2000 with the design by Hugo Herbelin and Pierre-Louis Curien of a symmetric calculus exhibiting deep dualities between the notion of programs and evaluation contexts and between the standard notions of call-by-name and call-by-value evaluation semantics.

3.4.3. Abstract machines

Abstract machines came as an intermediate evaluation device, between high-level programming languages and the computer microprocessor. The typical reference for call-by-value evaluation of λ-calculus is Landin's SECD machine [53] and Krivine's abstract machine for call-by-name evaluation [52], [51]. A typical abstract machine manipulates a state that consists of a program in some environment of bindings and some evaluation context traditionally encoded into a "stack".

3.4.4. Delimited control

Delimited control extends the expressiveness of control operators with effects: the fundamental result here is a completeness result by Filinski [44]: any side-effect expressible in monadic style (and this covers references, exceptions, states, dynamic bindings, ...) can be simulated in λ-calculus equipped with delimited control.

4. Application Domains

4.1. The impact of Coq

Coq is one of the 8 most used proof assistants in the world. In Europe, its main challengers are Isabelle (developed in Munich, Germany), HOL (developed in Cambridge, UK) and Mizar (developed in Białystok, Poland).

Coq is used in various research contexts and in a few industrial contexts. It is used in the context of formal mathematics at the University of Nijmegen (constructive algebra and analysis), INRIA Sophia-Antipolis (number theory and algebra), INRIA-MSR joint lab (group theory), the University of Nice (algebra). It is used in France in the context of computer science at INRIA-Rocquencourt (certified compilation), INRIA-Saclay (certification of imperative programs), LORIA, Strasbourg (certification of geometry algorithms). Outside France, it is used in the context of computer science e.g. at U. Penn (formalisation of programming languages semantics), Yale, Ottawa and Berkeley Universities (building of a certified platform for proof-carrying code), University of Princeton (certified compilation), AIST at Tokyo (certification of cryptographic protocols), Microsoft Research Cambridge (proof of imperative programs), ... In the industry, it is used by Gemalto and Trusted Logic (JavaCard formal model and commercial applets certification).

All in all, it is difficult to evaluate how much Coq is used. Two indicators are the readership of the textbook on Coq by Yves Bertot and Pierre Castéran [35] and the number of subscribers to the Coq-club mailing list. More than 1200 copies of the book have been sold. There has been a second printing , and a Chinese translation of the book has been published. There are around 600 subscribers to the mailing list. Coq is taught or used for teaching in many universities: Paris, Bordeaux, Lyon, Nice, Strasbourg, CNAM, Nottingham, Ottawa, U. Penn, Warsaw, Krakow, Princeton, Yale, Berkeley, Rosario in Argentina, ...

5. Software

5.1. Coq

Participants: Bruno Barras [TypiCal team, Saclay], Yves Bertot [Marelle team, Sophia], Frédéric Besson [Lande team, Rennes], Frédéric Blanqui [Formes team, Beijing], Pierre Boutillier, Pierre Corbineau [University Joseph Fourier, Grenoble], Pierre Courtieu [CNAM], Jean-Christophe Filliâtre [ProVal team, Saclay], Julien Forest [CNAM], Stéphane Glondu, Benjamin Grégoire [Marelle team, Sophia], Vincent Gross, Hugo Herbelin [correspondant], Stéphane Lescuyer [ProVal team, Saclay], Pierre Letouzey, Assia Mahboubi [TypiCal team, Saclay], Claude Marché [ProVal team, Saclay], Julien Narboux [University of Strasbourg], Jean-Marc Notin [TypiCal team, Saclay], Russell O'Connor [University of Nijmegen], Christine Paulin [Proval team, Saclay], Loïc Pottier [Marelle team, Sophia], Matthias Puech, Yann Régis-Gianas, Vincent Siles, Élie Soubiran, Matthieu Sozeau [ProVal team and Harvard University], Arnaud Spiwack [TypiCal team, Saclay], Pierre-Yves Strub [Formes team, Beijing], Laurent Théry [Marelle team, Sophia], Benjamin Werner [TypiCal team, Saclay].

5.1.1. Version 8.3

Version 8.3 was released in September 2010. It introduces a new decision procedure contributed by Loïc Pottier from the Marelle team and many improvements of existing features, including: an extended and generally more efficient module system (Élie Soubiran), more tactics (Hugo Herbelin, Pierre Letouzey, Matthieu Sozeau), more robust and efficient type classes (Matthieu Sozeau), more efficient and comprehensive libraries (revision of the library of finite sets and of the library of abstract arithmetic by Pierre Letouzey, revision of the sorting library by Hugo Herbelin).

5.1.1.1. Graphical interface

The integrated graphical interface of Coq (CoqIDE) is under revision: Vincent Gross implemented a fully operational new communication model based on process interaction rather than on threading (the reasons are: ability to support multiple Coq sessions, ability to interrupt Coq asynchronously, better robustness on non Unix-compliant operating systems, definition of a communication protocol reusable by other Coq interfaces).

5.1.1.2. Internal architecture of the Coq software

Pierre Letouzey has pursued his effort for a better build infrastructure for Coq. The Makefile has been quite improved, the new version being at the same time more robust and less cryptic. The alternative build system based on ocamlbuild, started last year, is now operational. In particular, it has allowed Pierre Letouzey to propose a fully-automated script for building the Windows versions of Coq, by the means of cross-compilation from a Linux environment. This way, Windows packages of Coq can be now obtained reliably in a matter of minutes.

Pierre Letouzey has also worked on providing more abstraction between Coq and the third-party parsing engine it uses: as a consequence, Coq is now able to use either camlp5 or Gallium's official camlp4, which makes Coq less dependent of the future evolutions of these parsing engines.

Pierre Letouzey has also continued his collaboration with Maxence Guesdon (MIRIAD) concerning his code analyser Oug, allowing to detect and remove thousands of lines of useless or obsolete code in Coq.

Many performance bottlenecks have also been investigated by Pierre Letouzey, for instance in the implementation of Coq universes, or in the new proof engine of Coq due to Arnaud Spiwack.

Yann Regis-Gianas optimised the hash-consing algorithm used in the kernel.

Thanks to the gtk-osx effort, Pierre Boutillier has provided a better integration of Coq in a MacOS X setting.

5.1.2. The Technological Development Action Coq

This "Action de Développement Technologique", which is coordinated by Hugo Herbelin, gathers the teams and individuals listed above.

Two national-level meetings have been organised as part of the ADT Coq. The first meeting has been organised in February 2010 at La Ciotat and gathered around 15 persons on the topic of equality and termination in Coq. The second meeting has been organised by πr^2 in October 2010 and gathered around 15 persons on the question of interfaces for proof assistants (Coq and Isabelle). The minutes of the meetings can be found at URL http://coq.inria.fr/adt.

The ADT Coq supported the second Coq Asian summer school that Jean-Pierre Jouannaud (Formes team) organised in August 2010 in Beijing. The ADT Coq also supported the second Coq workshop held in July in Edinburgh. Chaired by Yves Bertot, the workshop accepted 2 reviewed full papers and 9 informal contributed talks. More than 30 persons attended. The web page of the workshop is http://coq.inria.fr/coq-workshop/2009.

5.1.3. Modules in Coq

Élie Soubiran went on working on two improvements of the module system of Coq. In the first one, he splits the primitive notion of theory into two atomic constructions of name-space and structure. This leads to a more general system where one can define not only modules but also extensible name-spaces. In the second improvement, he deals with a new merging-of-structure combinator that subsumes inclusion and refinement. Such a combinator helps, among others, to handle "diamond like" modular constructions. As part of his PhD dissertation, a step-by-step reconstruction of the module system of Coq was also provided.

5.1.4. The Coq extraction

While Stéphane Glondu investigates the next generation of the extraction tool of Coq, which aims at being certified in Coq (see section "Certified Extraction" below), Pierre Letouzey is maintaining and improving the current implementation of the Coq extraction. In particular, since version 8.3, this extraction allows for a more complete translation of the Coq language towards Haskell : modules and functors are now handled in a way which isn't ideal, but works. Many practical aspects of the extraction have been reworked to provide a better user experience, in particular the code transformations done to optimise the extracted programs. Pierre Letouzey has also worked on better ways to allow replacing Coq code or axioms by external code during extraction, or ways to fine-tune the parts of Coq code that are kept or not during extraction.

5.1.5. Formalisation in Coq

Vincent Siles has extended his Coq formalisation of untyped PTS's to PTS's with judgemental equality, and proved that both presentations are equivalent. The whole formalisation can be found at http://www.lix.polytechnique.fr/~vsiles/coq/PTSATR.html.

Stéphane Glondu is working with Mehdi Dogguy on the formalisation in Coq of a type system for a timed asynchronous π-calculus that guarantees confluence.

Ronan Saillard is working on a mechanised formalisation of his master thesis work in Coq: he is extending standard simulation techniques to prove the correctness of compilers in order to handle cost annotating compilers.

Pierre Boutillier has worked on a library about lists that store their length in their type, as a good playground to study dependent pattern matching in Coq. The pattern-matching typing-rule in Coq provides a machinery to generalise upon the type dependencies of the matched term. The same machinery can sometimes be used to make the expected type fit the actual type in branches without explicitly rewriting. Using such mechanism to propagate type-constraints through case-analysis allows to write more natural programs whose reduction is never disturbed by reasoning about equality. Programs written in this setting are much easier to tackle in proofs afterwards.

Matthieu Sozeau has worked on a high-level interface for writing programs using dependent-pattern matching and helping reasoning about them, during his post-doc at Harvard. He is continuing work on this problem and seeking ways to integrate the above technique (i.e., Boutillier's work) inside his extension.

5.2. Pangolin

Participant: Yann Régis-Gianas.

In collaboration with Johannes Kanig (PhD student, LRI/INRIA Proval/UPS), Yann Régis-Gianas released a first alpha version of Pangolin, a back-end for Pangolin whose role is to encode higher-order theories in first-order logic. This tool is intended to serve as an interface between, on one side, Pangolin and Coq, and on the other side, external automatic provers.

5.3. Other software developments

Stéphane Glondu is involved in the maintenance of OCaml-related packages in Debian, which include OCaml itself, Coq, Ssreflect (an extension of Coq developed at INRIA-MSR joint center) and Ocsigen (a web framework developed at PPS). The Ubuntu distribution naturally benefits from this work. In collaboration with Stefano Zacchiroli, Mehdi Dogguy and Sylvain Le Gall, he developed a solution to enforce library linkability using inter-package relationships. This work has been presented at JFLA 2010, and the associated article has been selected for a special issue of Studia Informatica Universalis.

In collaboration with François Pottier (INRIA Gallium), Yann Régis-Gianas maintained Menhir, an LR parser generator for Objective Caml.

6. New Results

6.1. Proof-theoretical investigations

Participants: Pierre-Louis Curien, Hugo Herbelin, Danko Ilik, Guillaume Munch-Maccagnoni, Alexis Saurin, Vincent Siles, Noam Zeilberger.

6.1.1. Sequent calculus and Computational duality

6.1.1.1. Axiomatisation of call-by-need

In collaboration with Zena Ariola, Hugo Herbelin started a study of call-by-need λ-calculus with the objective to raise its study at the level of what is done for call-by-name and call-by-value. The common approach up to now was basically to only study call-by-need λ-calculus under the point of view of standard head reduction as a model of what happens in programming languages such as Haskell. Ultimately, the goal is to show that one can go much further and 1) to develop full reduction semantics and equational theories for call-by-need, 2) to develop a canonical extension of call-by-need with control 3) to show along the lines of the duality of computation that there exists a new λ-calculus dual to call-by-need worth to be studied too.

6.1.1.2. Focalisation

Alexis Saurin has investigated how the Focalisation theorem of linear logic can be proved by interactive means in Girard's Ludics (in Terui's Computational Ludics setting [63]). This resulted in a presentation in a Workshop in late September 2009, later published in the workshop post-proceedings [34] and in an extended version published in a MFPS conference 2010 [23]. Connections with algorithmic complexity are discussed since Focalisation in the framework of computational ludics can be connected with proof methods of the linear-speedup theorem [58].

Pierre-Louis Curien and Guillaume Munch have published their joint work [24] on a presentation of *focalising system L* that is well-suited to

- a smooth explanation of the move from a restricted syntax of cut-free classical proofs (the focalised ones) to a confluent calculus of focalised and polarised classical proofs with cuts;
- further investigations on quotienting proofs over irrelevant details of order of introduction of (negative) connectives, leading to clean links with Zeilberger's work [14] on the unity of duality and with ludics.

6.1.1.3. (Co)Inductive Types in Sequent Calculus

Herbelin, Sarnat and Siles worked towards a sequent calculus presentation of CIC – based on Herbelin's LJT and Cervesato and Pfenning's spine calculus – by starting with a simple type theory with inductive and coinductive types, pattern matching, and guarded least and greatest fixed-points (although it already seems clear that adding dependencies will require a dependent version of the cut rule). The formulation makes use of some ideas from contextual-modal type theory, which simplifies the presentation. The guard conditions, and accompanying normalisation proof, remain works-in-progress, although the ideas so far have resulted in presentations at DTP '10 and TYPES '10.

6.1.2. On the logical contents of delimited control

6.1.2.1. Delimited control and $\Lambda\mu$-calculus

In the continuation of his work with Silvia Ghilezan [6] on showing that Saurin's variant $\Lambda\mu$ [10] of Parigot's $\lambda\mu$-calculus [60] for classical logic was a canonical call-by-name version of Danvy-Filinski's call-by-value calculus of delimited control, Hugo Herbelin studied with Alexis Saurin and Silvia Ghilezan another variant of call-by-name calculus of delimited control. This is leading to a general paper on call-by-name and call-by-value delimited control.

Alexis Saurin published two articles in international conferences on this topics in 2010: the first one [26] introduced a hierarchy of calculi for delimited control in call-by-name (that is a CBN correspondent to Danvy-Filinski's CPS hierarchy) while the second one [27] established a standardisation theorem and characterises solvability for $\Lambda\mu$-calculus [10] [60] and introduces Böhm trees for $\Lambda\mu$-calculus. Those two works develop previous works by the author alone [10], [13] [20] or with Hugo Herbelin [6] [48].

Last October, he submitted a journal paper entitled *Böhm theorem and Böhm trees for $\Lambda\mu$-calculus*, invited for the special issue of FLOPS 2010, which subsumes [10][27].

Building on these, a joint work with Gaboardi gave rise to a short presentation to ITCS conference which is planned to be submitted to a conference early 2011.

6.1.2.2. Control delimiters, Markov's principle, and Double-negation shift

Hugo Herbelin discovered a relation between control delimiters and Markov's principle: in an intuitionistic logic extended with classical logic for Σ^0_1-formulae (i.e. for formulae that correspond to data-types) and a control delimiter, Markov's principle (i.e. the property that $\neg\neg\exists x\, A(x) \rightarrow \exists x\, A(x)$ for $A(x)$ decidable) becomes provable while still retaining the main property of intuitionism, namely that any proof of $\exists x\, A(x)$ contains a witness t such that $A(t)$ holds [25].

Hugo Herbelin and Danko Ilik discovered a connection between control delimiters and the Double-negation shift schema $((\forall x, \neg\neg A(x)) \rightarrow (\neg\neg(\forall x, A(x)))$, which is the main ingredient in providing an interpretation of the double-negation translation of the Axiom of Choice, and hence of consistency of Analysis.

Danko Ilik extended the logical system of [25], so that the Double-negation shift is indeed formally derivable, and he proved that this system also enjoys the disjunction and existence properties, i.e., is essentially intuitionistic. This work appears as a chapter of Ilik's recent thesis.

6.1.2.3. Delimited continuations, polarity and computational effects

Noam Zeilberger published and presented an article at the LICS 2010 conference in Edinburgh, titled "Polarity and the Logic of Delimited Continuations" [29]. The paper studied a generalisation of the classical interpretation of polarities, with the aim on the one hand of building a stronger connection between classical polarity (as in linear logic) and the "polarity-like" phenomena of intuitionistic systems (e.g., in Levy's Call-By-Push-Value [55], Watkins' Concurrent Logical Framework [64], etc.), and on the other hand of gaining a better understanding of delimited continuations and Filinski's representation theorem [44]. Since writing this article and concurrently, he has continued to work with Paul-André Melliès and Jonas Frey (PPS) on developing a more principled generalisation of *non-commutative* linear logic, with the ultimate goal of synthesising a logical account of abstract machines and computational effects. He presented some preliminary results from this work in December (at MSR Cambridge and at Oxford University) in a talk titled "Towards a non-commutative logic of effects".

Guillaume Munch-Maccagnoni investigated delimited control operators and delimited CPS translations from the point of view of linear logic. He provided a delimited control calculus which is polarised, in the sense of Girard's classical logic LC. This corresponds, in the operational semantics, to the coexistence of two dual modes of evaluation for expressions (strict and lazy). He shows how the CPS semantics of both call-by-value and call-by-name delimited control calculi factor through the polarised calculus. Many of the variants of delimited control calculi spawned as answers to the question of evaluation order in delimited control calculi, and thus the polarised calculus is one answer which is unifying to some degree.

He also shows that this polarised calculus decomposes down to linear logic (through polarised linear logic) like LC (using focusing and reversal). The only difference lies in the exponentials used in the construction: they are now indexed by formulae (which correspond to "annotations" in type-and-effect systems). The specific definition of these exponentials in linear logic forces a call-by-value semantics on the target of the CPS translation, which appears as a theoretical evidence in favour of Danvy and Filinski's original semantics for delimited control. As a by-product, he obtains factoring in three steps of CPS translations, which are also relevant to non-delimited CPS translations. These steps isolate distinct phenomena of CPS translations: the choice of a mode of evaluation, the coding of effects and bureaucratic artefacts that appear in CPS translations, like administrative redexes. These works are submitted to a conference.

Danko Ilik formalised in Coq the completeness proof for full intuitionistic logic (including disjunction and existential quantification) with respect to a newly introduced notion of model, similar to Kripke models, based on a dependently-typed continuations monad. In this way, he avoids the need of the Fan theorem from Veldman's proof of completeness with respect to standard Kripke semantics, and the need of delimited control operators that are essential for Danvy's normalisation-by-evaluation of lambda calculus with sums – the computational content of the formalised proof. This work is described in a chapter of Ilik's recent thesis [15].

6.1.3. The computational contents of completeness proofs

It is known from works on Normalisation by Evaluation (also known as semantic normalisation) that completeness proofs for intuitionistic logic along models such as Beth models are basically tools to map proofs of the meta-language into proofs of an object language. This scales to classical logic if one considers models such as boolean algebras. However, the computational content of completeness for classical logic with respect to the truth-values model is of a different nature. Krivine started to investigate this topic in 1996. Hugo Herbelin and Danko Ilik carried on this work and were able to extract a quite simple program which, when applied to a proof of validity of a first-order formula, produces a derivation of this formula, thus providing a computational content to one of the most central theorems of logic.

Previously, Hugo Herbelin, Danko Ilik, and Gyesik Lee gave a new kind of direct semantics for classical logic, similar to Kripke models, and proved constructively that it is sound and complete for first-order logic [19].

6.1.4. Substitutions and isomorphisms

During his one month visit, Martin Hofmann continued a work started with Pierre-Louis Curien earlier this year, when both were visiting Cambridge, on comparing their old respective categorical interpretations of Martin-Löf type theory back from the early nineties where they had cured a flaw of Seely's original interpretation in locally cartesian closed categories in somewhat symmetrical ways. Syntax has exact substitutions, while their categorical interpretation, in terms of pullbacks or fibrations, "implements" substitutions only up to isomorphism. One can then either change the model (strictification) [49], or modify the syntax (by introducing explicit substitutions and more importantly explicit coercions between types that are now only isomorphic) [4]. These approaches turn out to be nicely related through adjunctions in a suitable 2-categorical framework that has a conceptual interest of its own, which we largely owe to the third author of this collaboration, Richard Garner (University of Macquarie, Sydney). The results of this investigation are currently being written up.

6.2. Metatheory of Coq and beyond

Participants: Andreas Abel, Bruno Bernardo, Hugo Herbelin, Yann Régis-Gianas, Vincent Siles.

6.2.1. Normalisation

Pierre Boutillier has worked on extensions of the structural guard condition for fixpoints whose role is to ensure the termination of Coq programs. He strengthened the guard condition algorithm to ensure termination along all possible reduction strategies, instead of only one as it was before. He also extended it to support "commutative cuts", i.e. to support cuts interleaved with case analysis statements. While type preservation imposes to freeze commutative cuts, there is no reason to block them when it comes to termination checking. Finally, he investigated how to reformulate the guard condition efficiently and elegantly using ideas coming from abstract machines. Thanks to the commutative cuts trick, more dependently typed programs are now accepted by Coq.

6.2.2. Calculus of inductive constructions and typed equality

Hugo Herbelin and Vincent Siles extended their work on "full" Pure Type Systems [28] and showed that any Pure Type System, without any restrictions of functionality, fullness or normalisation, is equivalent to its typed counterpart. This not only closes a twenty years old open question on type theory, but it also allows to study extensions of Pure Type Systems, with subtyping or a stronger conversion for example, which would bring closer Coq's implementation to its theoretical description. This work can be found in Siles' PhD dissertation [16].

6.2.3. Implicit calculus of constructions, proof irrelevance

Bruno Bernardo is working on an Implicit Calculus of Constructions with dependent sums and with decidable type inference. In this calculus, all the explicit static information (types and proof objects) is transparent and does not affect the computational behaviour. Bruno Bernardo has already defined a formalism and studied an Implicit Calculus of Constructions [3]. The next step is to add Σ-types to the system. The syntax has already been extended. Subject reduction has been proven. The extension of Alexandre Miquel's models based on coherence spaces [57] is ongoing work that would lead to prove the consistency and the strong normalisation property of the system. This is joint work with Bruno Barras, researcher of the Typical team and PhD advisor of Bruno Bernardo.

In discussions with Bruno Bernardo and Bruno Barras, Andreas Abel investigated the relationship of proof irrelevance and implicit quantification in the CoC. This inspired a paper, written in October 2010, which is accepted for the FoSSaCS 2011 conference [22].

6.2.4. *Proofs of higher-order programs*

Jeffrey Sarnat and Noam Zeilberger have been investigating the two classical program transformations *continuation-passing-style* translation and *defunctionalisation* [61], from the point-of-view of their effect on the termination proofs of higher-order programs. Through the Curry-Howard correspondence, these termination proofs also correspond to consistency proofs of logics, and Sarnat and Zeilberger have explored a connection between defunctionalisation and Buchholz's idea of studying infinitary sequent calculi (e.g., with the ω-rule for arithmetic) by building *notation systems for infinitary derivations* [36]. The practical aim of these investigations is to develop a more systematic understanding of termination proofs, which eventually could result in a compiler from proof assistants with higher-order reasoning (such as Coq) to ones with only first-order reasoning (such as Twelf).

6.3. Coq as a functional programming language

Participants: Stéphane Glondu, Pierre Letouzey, Matthias Puech.

6.3.1. *Certified libraries*

In 2010, thanks to his current INRIA "délégation" period, Pierre Letouzey has continued a deep reform of some parts of the Standard Library of Coq, mainly the Numbers library of generic / efficient arithmetic. The idea is to take advantage of recent improvements of the Coq system in terms of modularity (Type Classes by Sozeau and better Modules by Soubiran) for providing more uniformity in the functions and properties about integers provided in the Standard Library. Currently Coq proposes three representations of natural numbers (unary, binary, and int31-based), and two representations of integers (binary and int31-based). With Pierre Letouzey's work this year, we are now sure that all these representations come along with at least the same set of basic functions (addition, multiplication, ... up to more advanced ones like gcd, square root, base-two logarithm or bitwise functions), and the same set of lemmas about these basic functions.

As a side effect of this work, the libraries BigN and BigZ of arbitrary large numbers based on blocks of int31 words have been improved, reducing the size of a macro-generated file, using instead an innovating approach based on Coq reduction at definition-time. The dependent-type aspects of these libraries have been put forward and exploited for a better organisation.

Pierre Letouzey is now working on providing an easy transition way to users from the earlier heterogeneous integer libraries to the new framework. He was also planning to work on tactics to guarantee that the same basic tactics (e.g. Presburger solver) are available whichever representation of numbers is used, but this part has not yet been done. This modular approach is also meant to be extended to other data-structures such as rational or real numbers.

For the moment, the abstract parts of the Numbers library rely heavily on advanced aspects of Coq module system, and is the result of a really fruitful interaction between Elie Soubiran and Pierre Letouzey, the latter benefiting from and/or suggesting improvements to the former. Pierre Letouzey had also several interactions with the group of Bas Spitters in Nijmegen about this question of modularity, trying to investigate which approach is best suited, modules, type-classes or maybe some combination of the two.

Matthieu Sozeau is developing the type-classes system taking into account input from the Nijmegen group and exploring its relation with logic programming.

6.3.2. *Certified extraction*

Stéphane Glondu continued his work on the internal extraction, which made him delve more into the implementation of Coq's kernel. This research track was quite time-consuming and didn't bring much results, so he focused back to the Coq-in-Coq formalisation, proving more results about extraction.

He presented his work at the workshop TYPES'10.

6.3.3. Incrementality in proof languages

Matthias Puech and Yann Régis-Gianas are currently working on the theoretical grounds of an incremental proof development and checking system. The traditional interaction with a proof-checker is a batch process. Coq (among others) refines this model by providing two forms of incrementality:

- a linear interaction mechanism (read-eval-print loop) for providing abbreviations and inductive definitions ;
- a set of gradual refinement tools (tactics) able to construct a term in a top-down fashion.

A more general approach to incrementality is being developed by means of a finer-grained analysis of dependencies. The approach developed is not restricted to the interaction with Coq, even if it is the targeted language, but is adaptable to virtually any typed formal language: the language and its dependencies are specified in a meta-language close to the Logical Framework λ_{LF}, in which subsequent versions of a development can be type-checked incrementally.

Partial results in this direction have been presented in [30]. Applications of this framework are: proof language for proof assistants, integrated development environments for proof or programming languages, typed version management systems.

6.3.4. Proofs of programs in Coq

As part of the CerCo european project, in collaboration with Roberto Amadio (PPS, Paris Diderot University), Nicolas Ayache, Ronan Saillard and Yann Régis-Gianas developed a prototype compiler for a large subset of the C language whose specificity is to annotate input source programs with information about the worst-case execution cost of their machine code. They conceived a proof technique to prove the correctness of such an annotating compiler. This work has been submitted to a conference [31].

7. Other Grants and Activities

7.1. National Actions

Alexis Saurin is member of the ANR LOGOI project (Logique et Géométrie de l'Interaction) and is co-ordinator of one of the tasks of the project, on computational models (interactive and quantum models of computations). The project kick-off meeting took place on the 30th of November.

In this prospect, Saurin aims at studying interactive models of computations, mainly along two directions: develop interpretations of logic and computation based on operator and von Neumann algebras (in Girard's GoI framework) and develop models of computation based on interaction for both functional programming and logic programming.

Pierre Letouzey is member of the ANR "Decert" project. The objective of the "Decert" project is to design an architecture for cooperating decision procedures, with a particular emphasis on fragments of arithmetic, including bounded and unbounded arithmetic over the integers and the reals, and on their combination with other theories for data structures such as lists, arrays or sets. To ensure trust in the architecture, the decision procedures will either be proved correct inside a proof assistant or produce proof witnesses allowing external checkers to verify the validity of their answers. In this prospect, Pierre Letouzey aims at integrating all results of this "Decert" project in the realm of the Coq proof assistant. Unfortunately, the implication of Pierre Letouzey in other tasks such that the reform of the arithmetical libraries of Coq has left little time for works related with Decert.

Several members of the team are active particpants of the ANR project CHOCO (Curry-Howard for Concurrency), which ends in April 2011. The national coordinator is Thoas Ehrhard (PPS), and the project garthers groups from IML (Marseille), LAMA (Chambéry), LIP (ENS Lyon), LIPN (Paris 13), and PPS. We also participate in the ANR project Deep Inference, coordinated by Lutz Strassburger (Parsifal team, INRIA Saclay)

7.2. Equipe Associée

Early October, the project-team submitted an "Équipe-associée" proposal (EA). Entitled SEMACODE (standing for Stratégies d'évaluation, Machines Abstraites et COntrôle Délimité) the EA has been selected early January. The EA gathers from the Inria side both people from the project-team (Boutillier, Curien, Glondu, Herbelin, Letouzey, Munch-Macagnoni, Saurin – coordinator – and Zeilberger) and Gaboardi from Focus INRIA team in Bologna. The foreign partners are located in the USA (Oregon) and in Serbia, namely with Ariola, Ghilezan and graduate students. The project aims in particular at developing collaborations on the formal investigation of evaluation strategies thanks to sequent calculus, formalizations of abstract machines and logical investigation of delimited control. All these themes are clearly situated in the scientific objectives of the project-team and will benefit from strengthened collaborations with the foreign partners. Concerning this last item, a workshop is planned in Novi Sad, Serbia, late May 2011. Moreover, it is planned to receive American graduates for internships.

7.3. EC projects

Yann Régis-Gianas is a participant of the EU-FP7 Certified Complexity project (CerCo). This European project started in February 2010 as a collaboration between Bologna university (Asperti, Coen), Edinburgh university (Pollack) and Paris Diderot university (Amadio, Régis-Gianas). The CerCo project aims at the construction of a formally verified complexity preserving compiler from a large subset of the C programming language to some typical micro-controller assembly language, of the kind traditionally used in embedded systems. Ronan Saillard's thesis and and Nicolas Ayache's postdoc are funded by this project.

8. Dissemination

8.1. Interaction with the scientific community

8.1.1. Collective responsibilities

Pierre-Louis Curien is deputy director of the Foundation "Sciences Mathématiques de Paris".

Hugo Herbelin coordinated the ADT Coq and the development of Coq.

8.1.2. Editorial activities

Pierre-Louis Curien is co-editor in chief of Mathematical Structures in Computer Science, and is an editor of Theoretical Computer Science and of Higher-Order and Symbolic Computation.

Pierre-Louis Curien is guest editor for a special issue of Logical Methods in Computer Science in connection with the conference TLCA 2009 (about 2/3 of the issue is already published online).

Pierre-Louis Curien wrote the preface of Girard's Festschrift, to appear in TCS (2011). This preface recalls some of the landmarks that allowed the cross fertilisation of program semantics and proof theory, starting around the mid eighties of the last century.

8.1.3. Program committees and organising committees

Alexis Saurin served on the program committee for the workshop Games and Logic for Programming Languages (GaLoP V, http://perso.ens-lyon.fr/olivier.laurent/galop10), a satellite event of ETAPS 2010 and will serve, in 2011, on the program committee of the International Conference on Functional Programming (ICFP 2011, http://www.icfpconference.org/icfp2011) and, for the second time, of the Workshop Games and Logic for Programming Languages (GaLoP VI, http://sites.google.com/site/galopws.

Pierre-Louis Curien was PC member of "Categorical logic", a satellite workshop of the joint conference MFCS / CSL 2010 in Brno, august 2010, and of Logic Colloquium, Paris, July 2010.

Pierre-Louis Curien is organising with Paul-André Melliès (PPS) the anniversary workshop GGJJ 2011, organised in honour of Gérard Berry and Jean-Jacques Lévy, Gérardmer, February 2011 (http://www.lri.fr/~conchon/gerardmer/index.html).

Hugo Herbelin was PC member of the workshop "Classical Logic and Computation" (satellite workshop of the joint conference MFCS / CSL 2010 in Brno), and of the Coq workshop (satellite workshop of FLOC 2010 in Edinburgh). He has also been invited in the PC of the conference TLCA 2011.

8.1.4. Jurys

In May 2010, Pierre-Louis Curien has been member of the "Comité de Sélection" for a professor position at the university of Chambéry.

8.1.5. Ph.D. and habilitation juries

Pierre-Louis Curien was a member of the jury of the thesis of Christine Tasson (Paris 7) and Pierre Clairambault (Paris 7). He was a reviewer for the habilitations of Olivier Laurent (Paris 7) and Lutz Strassburger (Paris 7).

Hugo Herbelin was a member of the jury of the thesis of Benoît Montagu (École Polytechnique) and was a reviewer for the Habilitation of Tristan Crolard.

8.2. Visits

8.2.1. Outbound

Andreas Abel visited Christophe Raffalli and Pierre Hyvernat (LAMA, Universite de Savoie, Chambery), from 8 to 12 February, working on a termination checker for PML and giving a talk on *Normalization by Evaluation for System F*. A similar talk was given at the meeting of the INRIA CORIAS project, headed by Gilles Dowek and Claude Kirchner, in Val d'Ajol, Vosges, France, which Andreas Abel joined from 8 to 12 March. From 24 to 30 March he attended and contributed the Agda Implementor's Meeting on Awaji Island in Japan. He presented his paper *Towards Normalization by Evaluation for the Calculus of Constructions*, which he prepared during his research visit to PI.R2, at the FLOPS 2010 Conference in Sendai, Japan, on 19 April 2010 [21].

Alexis Saurin visited CIS department of University of Oregon, USA, in February and March 2010. He visited RIMS, Kyoto, Japan, in April 2010 and the math department of University of Minas Gerais, Belo Horizonte, Brasil, in October 2010.

Noam Zeilberger visited the Programming, Logic, and Semantics Group at ITU Copenhagen for one week in May 2010. In December, he visited the Programming, Principles, and Tools group at Microsoft Research Cambridge, and the Oxford University Computing Laboratory.

Guillaume Munch-Maccagnoni visited Cambridge Computer Lab for one week in April 2010.

Pierre-Louis Curien visited Cambridge Computer Lab for three months (April through June 2010), for the second and last period of his Leverhume grant.

8.2.2. Inbound

Andreas Abel was a guest researcher in the PI.R2 team from 1 October 2009 to 31 March 2010, on leave from his assistant professorship at Ludwig-Maximilians-University, Munich, Germany.

Zena Ariola (University of Oregon) visited πr^2 for one week days in September 2010.

Carsten Schürmann (University of Copenhagen) visited πr^2 for one week in September 2010. At this occasion, he gave a one-day tutorial on the Twelf proof assistant.

Andrej Bauer (University of Ljubljana) visited πr^2 for three days in October 2010 and gave a talk on the Eff programming language prototype.

Ulrich Berger (University of Swansea) visited πr^2 and PPS for three days and gave a talk on computing with real numbers in Coq.

Olivier Danvy (University of Aarhus) visited πr^2 and PPS for one month and gave a talk on programming with continuations.

Randy Pollack (University of Edinburgh) visited πr^2 for two days and gave a talk on the formal representation of bindings on machine.

Martin Hofmann (University Ludwig Maximilian, Munich) visited πr^2 for one month (INRIA professor invitation programme).

8.3. Teaching

8.3.1. Supervision of Ph.D. and internships

Pierre Letouzey is currently the PhD advisor of Stéphane Glondu.

Hugo Herbelin has started supervising the PhD of Pierre Boutillier. Three of his students: Élie Soubiran, Danko Ilik and Vincent Siles have defended their thesis in 2010 [15], [16], [17].

Pierre-Louis Curien is the PhD advisor of Guillaume Munch (jointly with Thomas Ehrhard), and is also the supervisor of two students at PPS outside the πr^2 project (Stéphane Zimmermann, jointly with Thomas Ehrhard, and Alexis Goyet).

Yann Régis-Gianas supervises the PhD of Ronan Saillard.

8.3.2. Courses

Pierre-Louis Curien has taught for the second time a 48 hours proof theory course in the Master program "Logique Mathématique et Fondements de l'Informatique" at Paris 7. He gave a 6 hour proof theory course at Cambridge University (June 2010).

Pierre Boutillier, Stéphane Glondu, Matthias Puech, Ronan Saillard and Guillaume Munch are teaching assistants at University Paris Diderot-Paris 7. They teach this year Java programming (beginner / advanced), unix-system (beginner / advanced), algorithms (advanced) and proof assistant.

In February-March 2010, Alexis Saurin taught a 10H graduate course at University of Oregon CIS department on Proof Theory, Linear Logic and Proof Search while doing a research visit to UO, working with Zena Ariola. Moreover, during a visit to University of Minas Gerais in Belo Horizonte, Saurin gave one introductory lecture on λ-calculus and its connections with logic in a logic course to graduate students in philosophy.

Yann Régis-Gianas took part in the MPRI course entitled "Type systems". He gave a 12 hours course about generalised algebraic data types, higher-order Hoare logic and dependently typed programming.

Vincent Siles is a teaching assistant at Ecole Polytechnique. During 2010, he taught Java programming (beginner), intermediate OCaml programming and introduction to networking.

Bruno Bernardo was teaching as an ATER in Paris 7 until August 2010. He taught "Travaux dirigés" about *Web Site Design* (*Internet et Outils*). He also conceived and supervised a student project common to the *Script Languages* and *C Language* courses.

8.4. Participation in conferences and seminars

8.4.1. Invited talks

Pierre-Louis Curien was invited speaker at the Operads and Universal Algebra Conference, Tianjin, China, July 2010 (http://andromeda.rutgers.edu/~liguo/OUA10/operadua.html), and at the TYPES 2010 workshop.

8.4.2. Presentation of papers

Glondu: at JFLA 2010.

Herbelin: [25] at FLOC'10 (LICS) in Edinburgh and to the workshop FATPA'10 in Novi Sad.

Puech: [30] at the MIPS 2010 workshop.

Saurin: [27], [26] at FOSSACS 2010 and FLOPS 2010.

Siles: [28] at LICS 2010.

Zeilberger: [29] at LICS 2010.

Curien: [24] at IFIP TCS 2010.

8.4.3. Other presentations

Herbelin and Sarnat presented their joint work on the sequent calculus presentation of the Calculus of Inductive Constructions at DTP'10 workshop (part of FLOC'10) and TYPES'10, respectively.

Herbelin presented his work with Danko Ilik on the computational content of Gödel's completeness theorem at TYPES'10 in Warsaw and at the joint COST 0910 and Alpine Verification Meeting at Lugano.

Letouzey gave an invited presentation about extraction at Journées Francophones des Langages Applicatifs (JFLA, February, La Ciotat).

Andreas Abel also presented his research *On Irrelevance and Extraction in Type Theory* at the JFLA.

Letouzey gave two of the lessons in the one-week INRIA-EDF Summer School about Coq (June, Paris).

Ilik gave an invited talk entitled "Constructive Completeness Theorems and Delimited Control" at the Workshop on Constructive Aspects of Logic and Mathematics, organised by Japan Advanced Institute of Science and Technology (March, Kanazawa, Japan).

Sozeau gave a talk at TYPES'10 in Warsaw (October).

8.4.4. Attendance to conferences, workshops, schools,...

Oregon Programming Languages Summer School (Boutillier, Glondu, Puech, Saurin).

Journées Francophones des Langages Applicatifs, February, La Ciotat (Glondu, Letouzey).

Journées GEOCAL–LAC, March, Nice (Zeilberger, Munch-Maccagnoni).

Dagstuhl Seminar on Game Semantics and Verification, June (Zeilberger).

Types 2010 in Warsaw (Glondu, Herbelin, Sozeau, Curien).

Monthly meetings of the ANR project CHOCO (Curry-Howard for concurrency) at Lyon (Munch-Maccagnoni).

FLoC 2010 in Edinburgh (Glondu, Herbelin, Siles, Zeilberger).

Meetings of the Cerco european project in Bologna, Paris and Edinburgh (Ayache, Régis-Gianas, Saillard).

CICM 2010, July, Paris (Puech).

École Jeune Chercheurs en Informatique Mathématique 2010 in Chambéry, France (Munch-Maccagnoni, Siles).

Réalisabilité à Chambéry workshop (Munch-Maccagnoni).

École d'Hiver en Informatique Fondamentale, ENS Lyon (Munch-Maccagnoni).

8.4.5. Talks in seminars

Abel: Talk at the ProVal Seminar, Saclay, on 19 March, on *Type-Based Termination for Dependent Types*.

Munch-Maccagnoni: "Semantics Lunch", Cambridge Computer Lab (April), École des Jeunes Chercheurs en Informatique Mathématique (Chambéry), Séminaire LDP (IML, Marseille), CHoCo Seminar (Lyon), Séminaire Parsifal / Typical (LIX, Palaiseau).

Siles: Seminar at ENS Lyon, Plume Team.

Saurin: Seminars at University of Novi Sad, Novi Sad, Serbia, University of Oregon seminar, Eugene, USA, RIMS, Kyoto, Japan, University of Minas Gerais, Belo Horizonte, Brasil, CHOCO meeting, Lyon, France

Zeilberger: Workshop on Proofs and Meaning at the Maison des Sciences de l'Homme (March 2010), Dagstuhl Seminar on Game Semantics and Verification (June), Microsoft Research Cambridge (December), OASIS Seminar of the Oxford University Computing Laboratory (December).

Sozeau: Gallium Seminar at Rocquencourt (October).

Curien: séminaire de mathématiques et d'informatique de l'université de Mulhouse (décembre 2010, exposé sur les substitutions et les isomorphismes)

8.4.6. Groupe de travail Théorie des types et réalisabilité

This is one of the working groups of PPS, jointly organised by Hugo Herbelin and Paul-André Melliès, since September 2009. It is held weekly at the Antenne INRIA.

8.5. Other dissemination activities

Yann Régis-Gianas organised the "Fête de la Science" event for the computer science department of the Paris Diderot university, with some financial support of INRIA Paris-Rocquencourt communication services. Nicolas Ayache, Matthias Puech, Pierre Letouzey also took part in the animation.

Yann Régis-Gianas co-organised the "Journée Francilienne de Programmation", a programming contest between undergraduate students of three universities of Paris (UPD, UPMC, UPS).

Yann Régis-Gianas and Pierre Letouzey participated to the "Salon de la culture et des jeux mathématiques" as scientific orators for INRIA.

Yann Régis-Gianas gave several conferences about computer science in the high-schools Lycée Montaigne and Camille See.

Yann Régis-Gianas supervised the internships of four fourteen years old schoolboys in INRIA Paris-Rocquencourt.

9. Bibliography

Major publications by the team in recent years

[1] Z. M. Ariola, H. Herbelin, A. Sabry. *A Type-Theoretic Foundation of Continuations and Prompts*, in "Proceedings of the Ninth ACM SIGPLAN International Conference on Functional Programming (ICFP '04)", Snowbird,Utah, ACM, September 19-21 2004, p. 40–53.

[2] Z. M. Ariola, H. Herbelin, A. Sabry. *A Type-Theoretic Foundation of Delimited Continuations*, in "Higher Order and Symbolic Computation", 2007, http://dx.doi.org/10.1007/s10990-007-9006-0.

[3] B. Barras, B. Bernardo. *The Implicit Calculus of Constructions as a Programming Language with Dependent Types*, in "FoSSaCS", 2008, p. 365-379.

[4] P.-L. Curien. *Substitution up to isomorphism*, in "Fundamenta Informaticae", 1993, vol. 19, p. 51-85.

[5] P.-L. Curien, H. Herbelin. *The duality of computation*, in "Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00)", Montreal, Canada, SIGPLAN Notices 35(9), ACM, September 18-21 2000, p. 233–243 [*DOI :* 10.1145/351240.351262], http://hal.archives-ouvertes.fr/inria-00156377/en/.

[6] H. HERBELIN, S. GHILEZAN. *An Approach to Call-by-Name Delimited Continuations*, in "Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008", San Francisco, California, USA, G. C. NECULA, P. WADLER (editors), ACM, January 7-12 2008, p. 383-394.

[7] H. HERBELIN. *A Lambda-Calculus Structure Isomorphic to Gentzen-Style Sequent Calculus Structure*, in "Computer Science Logic, 8th International Workshop, CSL '94", Kazimierz, Poland, L. PACHOLSKI, J. TIURYN (editors), Lecture Notes in Computer Science, Springer, September 25-30 1995, vol. 933, p. 61–75.

[8] G. MUNCH-MACCAGNONI. *Focalisation and Classical Realisability*, in "Computer Science Logic '09", E. GRÄDEL, R. KAHLE (editors), Lecture Notes in Computer Science, Springer-Verlag, 2009, vol. 5771, p. 409–423.

[9] Y. RÉGIS-GIANAS, F. POTTIER. *A Hoare Logic for Call-by-Value Functional Programs*, in "Proceedings of the Ninth International Conference on Mathematics of Program Construction (MPC'08)", Lecture Notes in Computer Science, Springer, July 2008, vol. 5133, p. 305–335, http://gallium.inria.fr/~fpottier/publis/regis-gianas-pottier-hoarefp.ps.gz.

[10] A. SAURIN. *Separation with Streams in the $\Lambda\mu$-calculus*, in "Symposium on Logic in Computer Science (LICS 2005)", Chicago, IL, USA, Proceedings, IEEE Computer Society, 26-29 June 2005, p. 356-365.

[11] A. SAURIN. *On the Relations between the Syntactic Theories of $\lambda\mu$-calculi*, in "Computer Science Logic 2008", LNCS, Springer, 2008.

[12] A. SAURIN. *Typing Streams in the $\Lambda\mu$-calculus*, in "ACM Transactions on Computational Logic", 2009, to appear.

[13] A. SAURIN. *On the Relations between the Syntactic Theories of $\lambda\mu$-Calculi*, in "17th Annual Conference of the EACSL 17th EACSL Annual Conference on Computer Science Logic - CSL 2008", Bertinoro Italie, Lecture notes in computer science, Springer, 2008, vol. 5213, p. 154-168 [*DOI :* 10.1007/978-3-540-87531-4_13], http://hal.archives-ouvertes.fr/hal-00527930/en/.

[14] N. ZEILBERGER. *On the unity of duality*, in "Annals of Pure and Applied Logic", 2008, vol. 153(1), p. 66-96.

Publications of the year

Doctoral Dissertations and Habilitation Theses

[15] D. ILIK. *Preuves constructives de complétude et contrôle délimité*, Ecole Polytechnique X, October 2010, http://hal.inria.fr/tel-00529021/en.

[16] V. SILES. *Investigation on the typing of equality in type systems*, Ecole Polytechnique X, November 2010.

[17] É. SOUBIRAN. *Theory and namespace management for the Coq proof assistant*, Ecole Polytechnique X, September 2010.

Articles in International Peer-Reviewed Journal

[18] O. DELANDE, D. MILLER, A. SAURIN. *Proof and Refutation in MALL as a game*, in "Annals of Pure and Applied Logic", 2010, vol. 161, n^{o} 5, p. 654-672 [*DOI :* 10.1016/J.APAL.2009.07.017], http://hal.inria.fr/hal-00527922/en.

[19] D. ILIK, G. LEE, H. HERBELIN. *Kripke Models for Classical Logic*, in "Annals of Pure and Applied Logic", August 2010, vol. 161, n^{o} 11, p. 1367-1378 [*DOI :* 10.1016/J.APAL.2010.04.007], http://hal.inria.fr/inria-00371959/en.

[20] A. SAURIN. *Typing streams in the $\Lambda\mu$-calculus*, in "ACM Transactions on Computational Logic", 2010, vol. 11, n^{o} 4 [*DOI :* 10.1145/1805950.1805958], http://hal.archives-ouvertes.fr/hal-00527835/en/.

International Peer-Reviewed Conference/Proceedings

[21] A. ABEL. *Towards Normalization by Evaluation for the $\beta\eta$-Calculus of Constructions*, in "Functional and Logic Programming, 10th International Symposium, FLOPS 2010", Sendai, Japan, M. BLUME, N. KOBAYASHI, G. VIDAL (editors), Lecture Notes in Computer Science, Springer-Verlag, April 19-21 2010, vol. 6009, p. 224–239.

[22] A. ABEL. *Irrelevance in Type Theory with a Heterogeneous Equality Judgement*, in "Foundations of Software Science and Computational Structures, 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011", Saarbrücken, Germany, March 26 - April 3 2011, To appear..

[23] M. BASALDELLA, A. SAURIN, K. TERUI. *From Focalization of Logic to the Logic of Focalization*, in "Twenty-Sixth Conference on the Mathematical Foundations of Programming Semantics - MFPS XXVI", Canada Ottawa, 2010, p. 161-176 [*DOI :* 10.1016/J.ENTCS.2010.08.010], http://hal.inria.fr/hal-00527916/en.

[24] P.-L. CURIEN, G. MUNCH-MACCAGNONI. *The duality of computation under focus*, in "IFIP International Conference on Theoretical Computer Science", Australia Brisbane, Lecture Notes in Computer Science, Springer Verlag, 2010, http://hal.inria.fr/inria-00491236/en.

[25] H. HERBELIN. *An intuitionistic logic that proves Markov's principle*, in "Logic In Computer Science", United Kingdom Edinburgh, IEEE Computer Society, 2010, http://hal.inria.fr/inria-00481815/en.

[26] A. SAURIN. *A Hierarchy for Delimited Continuations in Call-by-Name*, in "13th International Conference on Software Science and Computational Structures - FOSSACS 2010", Cyprus Paphos, Lecture notes in computer science, Springer, 2010, vol. 6014, p. 374-388 [*DOI :* 10.1007/978-3-642-12032-9_26], http://hal.inria.fr/hal-00527925/en.

[27] A. SAURIN. *Standardization and Böhm Trees for $\Lambda\mu$-Calculus*, in "Tenth International Symposium on Functional and Logic Programming - FLOPS 2010", Japan Sendai, Lecture notes in computer science, Springer, 2010, vol. 6009, p. 134-149 [*DOI :* 10.1007/978-3-642-12251-4_11], http://hal.inria.fr/hal-00527926/en.

[28] V. SILES, H. HERBELIN. *Equality is typable in Semi-Full Pure Type Systems*, in "Logic In Computer Science - LICS 2010", United Kingdom Edinburgh, July 2010, 10 p., http://hal.inria.fr/inria-00496988/en.

[29] N. ZEILBERGER. *Polarity and the Logic of Delimited Continuations*, in "25th Annual IEEE Symposium on Logic in Computer Science (LICS 2010)", United Kingdom Edinburgh, 2010, p. 219 - 227 [*DOI :* 10.1109/LICS.2010.23], http://hal.inria.fr/hal-00548167/en.

Workshops without Proceedings

[30] M. PUECH, Y. RÉGIS-GIANAS. *Towards typed repositories of proofs*, in "Mathematically Intelligent Proof Search - MIPS 2010", France Paris, July 2010, http://hal.inria.fr/inria-00525874/en.

Other Publications

[31] R. M. AMADIO, N. AYACHE, Y. RÉGIS-GIANAS, R. SAILLARD. *Certifying cost annotations in compilers*, 2010, technical report, http://hal.inria.fr/hal-00524715/en.

[32] V. SILES, H. HERBELIN. *Pure Type System conversion is always typable*, 2010, en cours de soumission à JFP, http://hal.inria.fr/inria-00497177/en.

References in notes

[33] H. P. BARENDREGT. *The Lambda Calculus: Its Syntax and Semantics*, North Holland, Amsterdam, 1984.

[34] M. BASALDELLA, A. SAURIN, K. TERUI. *On the Meaning of Focalization*, in "(informal) Proceedings of Prelude Workshop", September 2009, http://www.pps.jussieu.fr/~saurin/Publi/BST-focalization-ludics.pdf.

[35] Y. BERTOT, P. CASTÉRAN. *Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions*, Springer, 2004.

[36] W. BUCHHOLZ. *Notation systems for infinitary derivations*, in "Archive for Mathematical Logic", 1991, vol. 30, p. 277–296.

[37] A. CHURCH. *A set of Postulates for the foundation of Logic*, in "Annals of Mathematics", 1932, vol. 2, p. 33, 346-366.

[38] T. COQ DEVELOPMENT TEAM. *The Coq Reference Manual, version 8.2*, September 2008, http://coq.inria.fr/doc.

[39] T. COQUAND. *Une théorie des Constructions*, University Paris 7, January 1985.

[40] T. COQUAND, G. HUET. *Constructions : A Higher Order Proof System for Mechanizing Mathematics*, in "EUROCAL'85", Linz, Lecture Notes in Computer Science, Springer Verlag, 1985, vol. 203.

[41] T. COQUAND, C. PAULIN-MOHRING. *Inductively defined types*, in "Proceedings of Colog'88", P. MARTIN-LÖF, G. MINTS (editors), Lecture Notes in Computer Science, Springer Verlag, 1990, vol. 417.

[42] H. B. CURRY, R. FEYS, W. CRAIG. *Combinatory Logic*, North-Holland, 1958, vol. 1, §9E.

[43] M. FELLEISEN, D. P. FRIEDMAN, E. KOHLBECKER, B. F. DUBA. *Reasoning with continuations*, in "First Symposium on Logic and Computer Science", 1986, p. 131-141.

[44] A. FILINSKI. *Representing Monads*, in "Conf. Record 21st ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, POPL'94", Portland, OR, USA, ACM Press, 17-21 Jan 1994, p. 446-457.

[45] G. GENTZEN. *Untersuchungen über das logische Schließen*, in "Mathematische Zeitschrift", 1935, vol. 39, p. 176–210,405–431.

[46] J.-Y. GIRARD. *Une extension de l'interpretation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types*, in "Second Scandinavian Logic Symposium", J. FENSTAD (editor), Studies in Logic and the Foundations of Mathematics, North Holland, 1971, n^{o} 63, p. 63-92.

[47] T. G. GRIFFIN. *The Formulae-as-Types Notion of Control*, in "Conf. Record 17th Annual ACM Symp. on Principles of Programming Languages, POPL '90", San Francisco, CA, USA, 17-19 Jan 1990, ACM Press, 1990, p. 47–57.

[48] H. HERBELIN, A. SAURIN. *??-calculus and ??-calculus: a Capital Difference*, F.: Theory of Computation/F.4: MATHEMATICAL LOGIC AND FORMAL LANGUAGES/F.4.1: Mathematical Logic/F.4.1.2: Lambda calculus and related systems, F.: Theory of Computation/F.4: MATHEMATICAL LOGIC AND FORMAL LANGUAGES/F.4.1: Mathematical Logic/F.4.1.7: Proof theory, F.: Theory of Computation/F.3: LOGICS AND MEANINGS OF PROGRAMS/F.3.3: Studies of Program Constructs/F.3.3.0: Control primitives, http://hal.inria.fr/inria-00524942/en/.

[49] M. HOFMANN. *On the Interpretation of Type Theory in Locally Cartesian Closed Categories*, in "Computer Science Logic (CSL'94)", Springer Lecture Notes in Computer Science 933, 1994, p. 427-441.

[50] W. A. HOWARD. *The formulae-as-types notion of constructions*, in "to H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism", Academic Press, 1980, Unpublished manuscript of 1969.

[51] J.-L. KRIVINE. *A call-by-name lambda-calculus machine*, in "Higher Order and Symbolic Computation", 2005.

[52] J.-L. KRIVINE. *Un interpréteur du lambda-calcul*, 1986, Unpublished.

[53] P. LANDIN. *The mechanical evaluation of expressions*, in "The Computer Journal", January 1964, vol. 6, n^{o} 4, p. 308–320.

[54] P. LANDIN. *A generalisation of jumps and labels*, UNIVAC Systems Programming Research, August 1965, n^{o} ECS-LFCS-88-66, Reprinted in Higher Order and Symbolic Computation, 11(2), 1998.

[55] P. B. LEVY. *Call-by-Push-Value: A Subsuming Paradigm*, in "TLCA", 1999, p. 228-242.

[56] P. MARTIN-LÖF. *A theory of types*, University of Stockholm, 1971, n^{o} 71-3.

[57] A. MIQUEL. *Le Calcul des Constructions implicite: syntaxe et sémantique*, Université Paris 7, December 2001.

[58] C. PAPADIMITRIOU. *Computational Complexity*, Addison Wesley, 1994.

[59] M. PARIGOT. *Free Deduction: An Analysis of "Computations" in Classical Logic.*, in "Logic Programming, Second Russian Conference on Logic Programming", St. Petersburg, Russia, A. VORONKOV (editor), Lecture Notes in Computer Science, Springer, September 11-16 1991, vol. 592, p. 361-380, http://dblp.uni-trier.de.

[60] M. PARIGOT. *Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction*, in "Logic Programming and Automated Reasoning: International Conference LPAR '92 Proceedings", St. Petersburg, Russia, Springer-Verlag, 1992, p. 190-201.

[61] J. C. REYNOLDS. *Definitional interpreters for higher-order programming languages*, in "ACM '72: Proceedings of the ACM annual conference", New York, NY, USA, ACM Press, 1972, p. 717–740.

[62] J. C. REYNOLDS. *Towards a theory of type structure*, in "Symposium on Programming", B. ROBINET (editor), Lecture Notes in Computer Science, Springer, 1974, vol. 19, p. 408-423.

[63] K. TERUI. *Computational Ludics*, in "Theoretical Computer Science", 2009, to appear.

[64] K. WATKINS, I. CERVESATO, F. PFENNING, D. WALKER. *A Concurrent Logical Framework I: Judgments and Properties*, Department of Computer Science, Carnegie Mellon University, 2002, n^o CMU-CS-02-101, Revised May 2003.

[65] N. DE BRUIJN. *AUTOMATH, a language for mathematics*, Technological University Eindhoven, November 1968, n^o 66-WSK-05.