IN PARTNERSHIP WITH:
CNRS

Université Denis Diderot (Paris 7)

Activity Report 2011

Project-Team PI.R2

Design, study and implementation of languages for proofs and programs

IN COLLABORATION WITH: Laboratoire Preuves, Programmes et Systèmes

RESEARCH CENTER
Paris - Rocquencourt

THEME
Programs, Verification and Proofs

Table of contents

Project-Team PI.R2

Keywords: Programming Languages, Interactive Theorem Proving, Type Systems, Proofs Of Programs, Proof Theory

πr^2 is a common project with CNRS and University Paris 7, within the laboratory "Preuves, Programmes et Systèmes" (which is itself joint between Paris 7 and CNRS – UMR 7126). The team has been created on January the 1st, 2009 and became an INRIA "équipe-projet" on July the 1st, 2009. Its explicit association with both University Paris 7 and CNRS is now very close to being finalised.

Beginning of the Team: 01/01/2011.

1. Members

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]

Engineer

Xavier Clerc [Ingénieur Coq]

PhD Students

Pierre Boutillier [Contrat doctoral université Paris 7]
Stéphane Glondu [Allocation couplée, Paris 7, until August 2011]
Lourdes del Carmen Gonzalez Huesca [Funding by the ANR PARAL-ITP, since December 2011]
Guillaume Munch-Maccagnoni [Contrat doctoral université Paris 7]
Pierre-Marie Pédrot [Contrat doctoral université Paris 7]
Matthias Puech [Cotutelle avec l'Université de Bologne, financement Ministère de la Recherche italien (MIUR)]

Post-Doctoral Fellows

Federico Aschieri [INRIA funding, since September 2011]
Nicolas Ayache [Postdoc of the european project "CerCo"]
Jeffrey Sarnat [INRIA funding, until June 2011]
Arnaud Spiwack [INRIA funding]
Noam Zeilberger [Postdoc of the Foundation SMP until October 2011, now postdoc at IMDEA Software institute in Madrid]

Administrative Assistant

Marine Meyer

2. Overall Objectives

2.1. Overall Objectives

The research conducted in πr^2 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 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 [30] 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 [27], then by Howard and de Bruijn at the end of the 60's [37], [49], 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 [23] stemmed out – a formalism that is both a logic and a programming language and that is at the source of the Coq system [48].

3.1.2. Towards the calculus of constructions

The λ-calculus, defined by Church [22], 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 [19].

For explaining the Curry-Howard correspondence, it is important to distinguish between intuitionistic and classical logic: following Brouwer at the beginning of the 20^{th} 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 [44].

In 1985, Coquand and Huet [23], [25] in the Formel team of INRIA-Rocquencourt explored an alternative approach based on Girard-Reynolds' system F [31], [47]. 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 [26] 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 OCaml 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 OCaml, 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

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.

3.3.1. 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.

3.3.1.1. 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.

3.3.1.2. 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 [34] 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 [43] and Reynolds [46] and started to be studied in an abstract way in the 80's by Felleisen *et al* [28], culminating in Parigot's $\lambda\mu$-calculus [45], 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 [42] and Krivine's abstract machine for call-by-name evaluation [41], [39]. 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 [29]: 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 [21] 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. http://coq.inria.frCOQ

Participants: Bruno Barras [TypiCal team, Saclay], Yves Bertot [Marelle team, Sophia], Frédéric Besson [Lande team, Rennes], Pierre Boutillier, Xavier Clerc [SED team], Pierre Corbineau [University Joseph Fourier, Grenoble], Pierre Courtieu [CNAM], 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], Julien Narboux [University of Strasbourg], Jean-Marc Notin [TypiCal team, Saclay], Christine Paulin [Proval team, Saclay], Loïc Pottier [Marelle team, Sophia], Matthias Puech, Yann Régis-Gianas, Vincent Siles, Élie Soubiran, Matthieu Sozeau, Arnaud Spiwack, Pierre-Yves Strub [Formes team, Beijing], Laurent Théry [Marelle team, Sophia], Benjamin Werner [TypiCal team, Saclay].

5.1.1. Version 8.4

Version 8.4 beta was released in December 2011. It introduces a new proof engine designed and implemented by Arnaud Spiwack and a new extensive modular library of arithmetic contributed by Pierre Letouzey. It also includes an extension of the underlying logic with "η-conversion" by Hugo Herbelin and "commutative-cuts compliant guard condition" by Pierre Boutillier, an extension of the pattern-matching compilation algorithm by Hugo Herbelin, an extension of the procedure of simplication of polynomial expressions by Loïc Pottier, a refinement of the type classes mechanism by Matthieu Sozeau, a new communication model by Vincent Gross for the graphical user interface CoqIDE, that Pierre Letouzey, Pierre Boutillier and Pierre-Marie Pédrot further extended.

Several users gracefully contributed improvements of various features (Tom Prince, Enrico Tassi, Daniel Grayson, Hendrik Tews, ...).

5.1.2. Graphical user interface

Pierre Letouzey has finalized the work initiated by Vincent Gross (former ADT engineer) concerning the CoqIDE user interface : CoqIDE and Coq are now separate unix processes, enhancing the reliability and improving the user experience.

5.1.3. Type inference, tactics, unification and type classes

Matthieu Sozeau corrected important issues with the unification algorithm and enhanced it to support universes. He improved the type-class implementation, adding support for forward-reasoning instances.

To improve the power of induction tactics, Hugo Herbelin added new heuristics for second-order pattern-matching based on ideas from Chung-Kil Hur's `Heq` plugin.

Pierre Letouzey extended the pattern-matching feature of the tactic language.

5.1.4. Internal architecture of the Coq software

Pierre Letouzey also initiated a large reorganization of the internal components of Coq, since these components are currently too much interdependent. This work aims at better isolating components and explicitating the interfaces between them. In addition to the initial goal of simplifying the compilation of Coq, having a clearer architecture is also expected to help new contributors when they discover and interact with this large and complex code-base. It also brings new prospects in direct communications between tools developed around Coq. This is a long-term effort that extends beyond the Coq 8.4 release.

Pierre Boutillier worked on the build system generator for Coq users contributions. It now handles correctly developments involving ML files. Files to build developments are also used by CoqIDE to infer the required arguments when it opens a file of a development.

5.1.5. Efficiency

Pierre Letouzey has pursued his effort concerning the improvement of many aspects of the internals of Coq. In particular, with Yann Régis-Gianas, he enabled a faster load of libraries by default, thanks to laziness, and also a better sharing of structures in memory (via better hash-consing), with lower memory footprint and some speedup as visible result. Many bugs have also been addressed.

Starting from September, Xavier Clerc has worked on the codebase in order to profile typical executions. Some hotspots were identified, most notably in comparison functions: some minor modifications led to a gain of a few percents on average. Some tests led to envision the use of a Coq-specialized version of comparison functions, superseding the generic OCaml ones.

5.1.6. General maintenance

Hugo Herbelin, Pierre Letouzey, Pierre Boutillier, Stéphane Glondu and Matthieu Sozeau worked on the general maintenance of the system.

5.1.7. Development Action

A new "Action de Développement Technologique" about Coq has started September 2011. It gathers the πr^2 team, the Marelle team and the CPR team from CNAM, Hugo Herbelin acting as the coordinator. It supports visits and meetings between developers and aims at strengthening the community of Coq users and contributors.

5.1.8. Formalisation in Coq

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.

5.2. Pangolin

Participant: Yann Régis-Gianas.

Yann Régis-Gianas maintained a prototype version of Pangolin. He used it to prove concrete complexity bounds for a set of functional programs using the method described in his FOPARA 2011 paper [16].

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 François Pottier (INRIA Gallium), Yann Régis-Gianas maintained Menhir, an LR parser generator for OCaml.

6. New Results

6.1. Proof-theoretical investigations

Participants: Federico Aschieri, Pierre Boutillier, Pierre-Louis Curien, Hugo Herbelin, Danko Ilik, Guillaume Munch-Maccagnoni, Pierre-Marie Pédrot, Alexis Saurin, Arnaud Spiwack, Noam Zeilberger.

6.1.1. Sequent calculus and Computational duality

Thunks and duality. Guillaume Munch-Maccagnoni investigated a notion dual to the thunks of call-by-value lambda-calculus which allows to simulate call-by-value in call-by-name. He started to investigate how this structure arises in many models of call-by-name lambda-calculus, how it could explain various phenomena such as storage operators, and how it could relate to features of actual programming languages.

Categorical semantics. Guillaume Munch-Maccagnoni started a collaboration with Marcelo Fiore with the aim of understanding structures behind sequent calculus (of system L in particular) and polarisation (as found in Girard's classical logic). The goal is to draw links with algebraic and/or computing structures in the unifying language of category theory. This work is also in collaboration with Pierre-Louis Curien.

Noam Zeilberger has continued to work with Paul-André Melliès (PPS) on developing a categorical framework for better understanding contexts and inference rules in proof theory and type theory, with the aim of achieving an integration with the theory of side-effects in programmming languages. He presented some results from this work in March at the European Workshop on Computational Effects.

Polarised Peano arithmetic. Guillaume Munch-Maccagnoni extended polarised classical logic and polarised classical realisability to predicate calculus and to Peano arithmetic. This decomposes and simplifies technical artefacts found in call-by-name classical realisability, and sheds a new light on witness extraction from proofs of Σ formulae.

(Co)Inductive Types in Sequent Calculus. Hugo Herbelin and Jeffrey Sarnat continued their work towards a sequent calculus presentation of a simply-typed fragment of CIC that has inductive and coinductive types, as formalized using recursion operators and a guard condition. Some progress was made on the formalization of the guard condition and normalization proof, but both remained unfinished as of the end of Sarnat's postdoc in June.

Classical call-by-need and the duality of computation. In a collaboration with Zena Ariola (and especially after a 3-week visit by Alexis Saurin in Oregon early 2011), Zena Ariola, Hugo Herbelin and Alexis Saurin have presented the call-by-need strategy in the framework of *the duality of computation*, that is a sequent calculus approach to call-by-need, and they extended call-by-need from minimal logic to classical logic which allowed to integrate smoothly control operators, resulting in particular in a call-by-need $\lambda\mu$-calculus. Moreover, the duality principles involved in such a framework unveiled a new calculus, dual to the usual call-by-need but which is distinct from call-by-name, call-by-value as well as the usual call-by-need. These results were presented at TLCA 2011.

Pursuing the previous collaboration, the three previous authors, together with Paul Downen and Keiko Nakata, studied abstract amchines for this classical call-by-need calculus.

6.1.2. Linear dependent types

Arnaud Spiwack started investigating dependent types variants of linear sequent calculus based on Curien & Herbelin's $\mu\tilde{\mu}$. The goal is to study what kind of set theory arises from such a linear type theory when following the tradition of intuitionistic type theory of defining a set as a type equipped with a relation (this construction is also known as *setoid*). Sets defined in this manner in Coq gives rise to a quasitopos (as proved in Arnaud Spiwack's PhD thesis) which makes for a reasonable approximation of usual mathematics, however

"linear sets" should be quite different and may support some unorthodox mathematics. To have an appropriate theory of linear sets seems to require a fairly rich linear type theory in particular one that supports so called *strong elimination* (the type theoretic equivalent to induction). So far, if extending $\mu\widetilde{\mu}$ to dependently typed linear logic has been achieved, strong elimination proves to be harder to coin.

Pierre-Marie Pédrot just started his PhD on this same general topic. While linear logic appeared as an operational decomposition of intuitionistic logic, dependent types are conversely an essential enrichment of the latter, as they permit to constructively formalize important parts of mathematics. Even though it is nowadays pervasively thought that we should mix them, actually it does seem that nobody seriously tried hitherto. This subject is quite vast, and both sides may mutually enhance the understanding of one another. From the linear side, linear logic is seriously lacking any satisfying syntax at all; and worse, it seems more prone to solely describe computational behavior instead of truly formalizing mathematics, for its very lack of richer types. From the dependent side, usual dependent type systems are based upon PTS, which, being a plain enrichment of basic lambda-calculus, are intrinsically call-by-name structures. Hence, a linear decomposition inspired from polarization techniques may permit a better analysis of the inner, yet-to-be discovered gears of PTS. One may even hope to include non-intuitionistic effects therein. Furthermore, practical systems used today (Coq, for example) come bundled with additional constructs, such as inductives, whose understanding with respect to models is still highly incomplete. One could expect linear logic to shed a new light upon these issues. This thesis stems from preliminary work of Pierre-Marie Pédrot (during his M2 internship) inspired by models of GoI and other closely related results from Girard, which suggest a natural way to integrate dependency into them.

6.1.3. Proving with side-effects

Axiom of dependent choice. Hugo Herbelin showed that classical arithmetic in finite types extended with strong elimination of existential quantification proves the axiom of dependent choice. To get classical logic and choice together without being inconsistent is made possible first by constraining strong elimination of existential quantification to proofs that are essentially intuitionistic and second to turn countable universal quantification into an infinite conjunction of classical proofs evaluated along a call-by-need evaluation strategy so as to extract of them intuitionistic contents that complies to the intuitionistic constraint put on strong elimination of existential quantification. This work has been presented at the TYPES conference.

Memory assignment, forcing and delimited control. Hugo Herbelin investigated how to extend his work on intuitionistically proving Markov's principle [35] and the work of Danko Ilik on intuitionistically proving the double negation shift (i.e. $\forall x\,\neg\neg A \to \neg\neg\forall x\,A$) [38] to other kind of effects. In particular, memory assignment is related to Cohen's forcing as emphasized by Krivine [40] and as the observation that Cohen's translation of formula P into $\forall y \leq x \exists z \leq y\, P(z)$ is similar to a state-passing-style transformation of type P into $S \to S \times P$.

Hugo Herbelin then designed a logical formalism with memory assignment that allows to *prove* in direct-style any statement provable using the forcing method, the same way as logic extended with control operators allows to support direct-style classical reasoning. Thanks to the use of delimiters over "small" formulas similar to the notation of Σ^0_1-formulas in arithmetic, the whole framework remains intuitionistic, in the sense that it satisfies the disjunction and existence property.

Two typical applications of proving with side-effects are global-memory proofs of the axiom of countable choice and an enumeration-free proof of Gödel's completeness theorem.

The main ideas of this research program have been presented at the Geocal-Lac meeting of the GDR IM.

6.1.4. Delimited control

Delimited control and infinitary/stream calculi

During his summer intership, Paul Downen studied with Alexis Saurin some infinitary λ-calculi and infinitary rewriting and in particular a proposal by Ketema, Blom, Aoto and Simonsen which allowes to consider transfinitely deep terms. The proposed calculus presented several defects on which Paul Downen studies focused. Some of these defects were corrected but the work is still on-going.

In a collaboration with Marco Gaboardi and Koji Nakazawa, Alexis Saurin has been studying how to turn the $\Lambda\mu$-calculus into a truly stream-based calculus. This involed enlarging the syntactical catergory for streams, defining a type system and comparing with other proposals for computing on streams.

Alexis Saurin also developed previous results on $\Lambda\mu$-calculus in a paper currently under final revision for publication in TCS.

PTS and delimited control Following Danvy-Filinski' simply-typed system for a λ-calculus with delimited control, Hugo Herbelin and Pierre Boutillier have defined a set of rules for pure type systems with control operator. The work relies on the CPS used to encode them in standard Pure Type Systems and involves extra type annotations. Nevertheless, it seems to be more general than previous attempts to build classical PTS [20]. It has been presented at the workshop TPDC (Theory and Practice of Delimited Continuations) in Novi Sad and an article is under preparation.

6.1.5. Interactive Realizability

Thanks to the Curry-Howard correspondence for classical logic, it is possible to extract programs from classical proofs. These programs use control operators as a way to implement backtracking and processes of intelligent learning by trial and error. Unfortunately, such programs are often hard to write, difficult to understand and are *very inefficient*: every time a program backtracks, it forgets way too much information. This state of thing is due to a poor understanding and control of the backtracking mechanism that interprets classical proofs. In order to write down more efficient programs, it is necessary to describe exactly: a) what the programs learn, b) how the knowledge of programs varies during the execution.

A first step towards this goal is the theory of Interactive Realizability, a semantics for intuitionistic Arithmetic with excluded middle over semi-decidable predicates. It is based on a notion of state, which describes the knowledge of programs coming from a classical proof, and explains how the knowledge evolves during computation.

Federico Aschieri is working in two directions. First he is extending this realizability semantics to second-order intuitionistic Arithmetic with the same excluded middle over semi-decidable predicates. He has also discovered a new state passing style transformation, which allows to implement in System F efficient programs, which backtrack at the right point and do not forget anything when backtracking. He is also investigating an interesting relation with the forcing semantics: it seems that his transformation is a very direct, new constructive formulation of forcing.

Secondly, in collaboration with Berardi, he is also extending Interactive Realizability to first-order Arithmetic, with full excluded middle. This work promises to provide a significantly finer description of the learning mechanism that interprets classical proofs.

6.1.6. Substitutions and isomorphisms

Pierre-Louis Curien extended his collaboration with Martin Hofmann (Univ. of Munich) and Richard Garner (MacQuarie University, Sydney), started in 2010, to the point where the picture sought and announced in the report of last year turned out to be a bit less idyllic than expected. Let us recall the question addressed. We wanted to compare precisely two ways of giving a categorical interpretations of Martin-Löf type theory, both overcoming the following mismatch: 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) [36], or modify the syntax (by introducing explicit substitutions and more importantly explicit coercions between types that are now only isomorphic) [2]. These approaches turn out to be related through adjunctions in a suitable 2-categorical framework that has a conceptual interest of its own. But these adjunction do not fit entirely together, as we found out early 2011: One is base-dependent, and the other not. So we cannot put them directly aside to get the nice conceptual picture we hoped for. Still, our initial goal of expressing the interpretations in terms of on another can be attained, but this remains to be worked out in detail.

6.1.7. Miscellanea

During his three months visit in Beijing, Pierre-Louis Curien worked on the relations between Rewriting theory and the theory of Gröbner bases and other bases like Janet bases or involutive bases that have been introduced in computer algebra. These comparisons shed some light on the classification of various completion techniques for rewriting systems (a completion turns a rewriting system into an equivalent locally confluent one). This is work in progress.

6.2. Metatheory of Coq and beyond

Participants: Pierre Boutillier, Hugo Herbelin, Yann Régis-Gianas, Jeffrey Sarnat, Vincent Siles, Matthieu Sozeau, Noam Zeilberger.

From the work he has done last year on the Coq termination checker. Pierre Boutillier wrote an article to describe formally the exact new Coq implementation of a structural guard condition that handles commutative cuts.

6.2.1. Calculus of inductive constructions and typed equality

The work of Hugo Herbelin and Vincent Siles on the equivalence of Pure Type Systems with typed or untyped equality has been accepted for publication [11].

6.2.2. Proofs of higher-order programs

Jeffrey Sarnat and Noam Zeilberger continued to investigate the classical program transformations of *continuation-passing-style* translation and *defunctionalisation* [46], from the point-of-view of their effect on the termination proofs of higher-order programs. 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)

6.2.3. Unification

Matthieu Sozeau and Hugo Herbelin worked on improving the unification algorithm of Coq, making it more predictable and resolving important soundness issues (e.g. type-checking, dealing with universes). In collaboration with Beta Ziliani at (PhD student at MPI Sarbrucken) and Aleksandar Nanevski (Resarcher at IMDEA Madrid), Matthieu Sozeau started a project to formalize (on paper) the improved unification algorithm, taking into account advanced features such as canonical structures and type classes. This will give a detailed view of the system to power users like [33] and improve the system to handle the delicate usage patterns developed in the Mathematical Components team at MSR to scale Coq to large formalizations.

6.3. Coq as a functional programming language

Participants: Stéphane Glondu, Pierre Letouzey, Matthias Puech, Matthieu Sozeau.

6.3.1. Type-classes and libraries

Matthieu Sozeau has worked with members of the Foundations team at Nijmegen on enhancing the implementation of type-classes to suit the needs of the development of the MathClasses formalization of abstract algebra (part of the ForMath EU project). This gave rise to an experimental implementation of forward reasoning for instance resolution. Concurrently, he started a collaboration with Jael Kriener (Univ. of Canterbury, who visited for a week in November) to adapt techniques from logic programming to the system, including a determinacy analysis that would give better control of the system when building large hierarchies of structures (as in the MathClasses library).

Pierre Castéran from INRIA Bordeaux and Matthieu Sozeau are developing a tutorial on the use of type-classes that will be the basis of an invited tutorial given by M. Sozeau at the JFLA conference in February 2012. It will be published as part of the new version of the Coq'Art book.

6.3.2. Equations

Matthieu Sozeau has continued the development of the Equations package to build and reason on dependently-typed programs. He reworked the internals of the tool for a more efficient implementation and extended it to handle recursion on arbitrary inductive families. He developed a sizable example of the use of the tool for proving the metatheory of the LF system. The plugin will be relased with the upcoming 8.4 version of Coq. A journal article presenting the tool and its usage is in preparation.

6.3.3. Recursive definitions in proof assistants

Together with Ana Bove and Alexander Krauss, Matthieu Sozeau has written a survey article on tools and methods to build and reason on recursive functions in proof assistants based on type theory. The survey compares the relative strenghts and weaknesses of various formalization methods available in constructive type theories like Agda or Coq and classical systems like HOL. The article is still under review.

6.3.4. Dependent pattern-matching

Hugo Herbelin and Pierre Boutillier worked on a new simulation of Agda's style dependent pattern-matching [24] in the Calculus of Inductive Constructions which, on the contrary of Goguen *et al*'s simulation [32], does not rely on any explicit notion of dependent equality. The simulation relies on a systematic re-generalization of a given matching over the next pattern to matching and over a notion of matching narrowed by type constraints formalized by Pierre Boutillier [14]. The former itself relies on Pierre Boutillier's master thesis on making the guard condition traversing blocked commutative cuts. The simulation has been implemented in Coq 8.4 but without narrowing on subpatterns yet.

6.3.5. Modularized arithmetical libraries

Pierre Letouzey has integrated in Coq a deep reform of some parts of its Standard Library, mainly the Numbers library of generic / efficient arithmetic. This reform is part of the the version Coq 8.4. 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. We now have a base of functions and lemmas which is statically guaranteed to be coherent from one numerical datatype representation to the other, allowing the user to choose more easily between these representations, according to the user's need in term of simplicity or efficiency. These modernized libraries are also one of the first large-scale experimentations with many recent features of modules and type-classes, and have helped maturing them and established new usage guidelines. The transition from previous versions of these libraries should be relatively transparent thanks to a compatibility layer.

6.3.6. Certified extraction

Stéphane Glondu continued his work on extraction in the Coq-in-Coq formalisation. He proved that reductions in the source language can be simulated in the target language (the converse had been proven two years before). This proof highlighted a critical bug in the actual implementation of Coq that has been solved by Pierre Letouzey.

The formalised extraction is only a step of the actual implementation of Coq: replacing logical subterms by an inert constant. Stéphane Glondu considers this work done, even though some parts have been admitted. He worked on how the formalisation could be redesigned in order to avoid currently admitted lemmas and to allow a better integration with the existing Coq system.

6.3.7. Incrementality in proof languages

Matthias Puech and Yann Régis-Gianas worked on incremental type checking. This preliminary work will be presented during a contributed talk at TLDI 2012 [15]. It sets the grounds of an incremental proof development and checking system, by means of a representation language for repositories of proofs and proof changes.

The traditional interaction with a proof-checker is a batch process. Coq (among others) refines this model by providing a read-eval print loop with a global undo system, implemented in an ad-hoc way. A more general approach to incrementality is being developed by means of a finer-grained analysis of dependencies. The approach developed is adaptable to any typed formal language: the language is specified in a meta-language close to the Edimburgh Logical Framework, in which subsequent versions of a development can be type-checked incrementally. Applications of this framework are: proof language for proof assistants, integrated development environments for proof or programming languages, typed version control systems.

6.3.8. Proofs of programs in Coq

As part of the CerCo european project, in collaboration with Roberto Amadio (PPS, Paris Diderot University), Nicolas Ayache and Yann Régis-Gianas maintained 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. Yann Régis-Gianas started a mechanized version of the proof technique used to prove the correctness of such an annotating compiler.

Yann Régis-Gianas developed another compiler for Core ML that uses a generalization of CerCo technique in order to obtain certified worst case execution time bounds on functional programs. This compiler produces proof obligations in Coq. The corresponding paper will be published in january 2012 in the proceedings of FOPARA 2011. It is available as a technical report [16].

7. Partnerships and Cooperations

7.1. National Initiatives

Matthieu Sozeau is member of the ANR Typex project (Types and certification for XML) and is coordinator of one of the tasks of the project on formalization and certification of XML tools. The project will kick-off on January 8th, 2012 and is a joint project with LRI, PPS and INRIA Grenoble.

Matthieu Sozeau, Hugo Herbelin and Yann Régis-Gianas are members of the ANR Paral-ITP started November 2011. Paral-ITP is about preparing the Coq and Isabelle interactive theorem provers to a new generation of user interfaces thanks to massive parallelism and incremental type-checking.

Hugo Herbelin is the coordinator of the PPS site for the ANR Récré accepted in 2011 and starting January 2012. Pierre Letouzey is member of the ANR Récré. Récré is about realisability and rewriting, with applications to proving with side-effects and concurrency.

7.2. European Initiatives

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. Nicolas Ayache's postdoc is funded by this project.

Hugo Herbelin, Pierre Letouzey and Matthieu Sozeau submitted a proposal for a PHC Van Gogh with the university of Nijmegen in the Netherlands. This proposal is about the mathematical libraries of Coq, type classes, and the homotopic interpretation of equality in the Calculus of Inductive Constructions.

Hugo Herbelin is the participant of a submitted proposal for a PHC Pavle Savić with the university of Novi Sad in Serbia, the mathematical institute of Belgrade, ENS Lyon and the university of Turin. This proposal is on a general thematic logic and types. Hugo Herbelin is also the participant of a submitted proposal for another PHC Pavle Savić with the university of Belgrade and the university of Strasbourg. This proposal is about automated deduction and formal geometry.

7.3. International Initiatives

7.3.1. INRIA Associate Teams

7.3.1.1. SEMACODE

Title: Proof theory and functional programming languages

INRIA principal investigator: Alexis SAURIN

International Partner:

Institution: University of Oregon (United States)

Laboratory: Computer and Information Science Department

Researcher: Zena ARIOLA

International Partner:

Institution: University of Novi Sad

Laboratory: Faculty of Engineering

Researcher: Silvia GHILEZAN

Duration: 2011 - 2013

See also: http://www.pps.jussieu.fr/~saurin/EA-SEMACODE

Cross-fertilization between logic and programming languages theory is at the root of many striking developments in programming concepts as well as tools for formal analysis of programs. Our associated team project aims at gathering senior and young researchers from both sites in order to put a joint research effort on the following research themes:formalizing particular evaluation strategies of functional languages based logical techniques coming from sequent calculi. More specifically, we shall be interested in incorporating control operator directly in call-by-need and in developing a uniform framework for call-by-value and call-by-name calculi with delimited controlinvestigating (delimited) control operators, in particular to unveil the logical interpretation of delimited control (that is its logical counter-part with respect to Curry-Howard correspondence), and developing; connections between delimited control and stream calculi;developing the logical content of realistic abstract machines and associated formal analysis tools for realistic abstract machines building on Curien-Herbelin lambda-bar calculi. The project will gather PiR2 expertise in proof theory and in the logical foundations of functional programming languages, the expertise of the oregonian group on call-by-need evaluation and delimited control as well as respective crucial inputs of Gaboardi and Ghilezan on stream calculi, delimited control, semantics and type theory. The project will in particular allow to have the INRIA and American students and post-docs involved in the project (7 out of 13 people involved) to travel between both sites and to organize joint workshops (one such workshop is planned in June 2011).

7.3.2. Visits of International Scientists

Olivier Danvy (University of Aarhus) visited πr^2 and PPS for one month and gave a talk on call-by-need abstract machines.

Beta Ziliani (MPI, Sarbrucken) visited πr^2 and PPS for a week in november and gave a talk on ad-hoc proof automation.

Danko Ilik visited πr^2 and PPS for one week and gave a talk on normalization by evaluation for delimited control.

Silvia Ghilezan (University of Novi Sad) visited πr^2 and PPS for one week and worked with Alexis Saurin and Hugo Herbelin on the classification of several calculi of delimited control.

Zerna Ariola (University of Oregon) visited πr^2 for one week and worked with Alexis Saurin and Hugo Herbelin on the definition of abstract amchines for classical call-by-need.

Keiko Nakata (University of Tallin) visited πr^2 and worked with Hugo Herbelin on recursive definitions and control operators in call-by-need λ-calculus.

Gyesik Lee visited πr^2 for one week and worked with Hugo Herbelin on the formal representation of binders in Coq and on representing primitive recursive arithmetic in Coq.

7.3.2.1. Internship

Paul Downen and Luke Maurer (University of Oregon) spent two months in the team during the summer, working with Alexis Saurin and Hugo Herbelin. Paul Downen studied calculi for multi-prompt, the derivation of abstract machines as well as infinitary λ-calculi. Luke Maurer studied Coq and did some formalization in Coq and studied Zeilberger's polarized approach to delimited control as well as connections with $\Lambda\mu$-calculus.

7.3.3. Visits abroad of members of the team

Matthieu Sozeau visited Ana Bove in Gothenburg for a week in January and gave a talk on Equations and dependent pattern-matching. They worked on a joint paper on tools and methods for recursion in type theory.

Matthieu Sozeau visited Alexandar Nanvesky at IMDEA Madrid from 19th to 23rd october and gave a talk on Type Classes and unification. They worked on the unification algorithm of Coq.

Guillaume Munch-Maccagnoni visited the Programming, Logic, and Semantics Group at the University of Cambridge from March to June. He is grateful to the Fondation Sciences Mathematiques de Paris which provided the funding.

Pierre-Louis Curien visited the University of Tsinghua in Beijing for three months, from March to May (funded by the Professor Group Chair of the Software School of this university). He was hosted by Gu Ming and Jean-Pierre- Jouannaud.

8. Dissemination

8.1. Animation of the scientific community

8.1.1. Collective responsibilities

Pierre-Louis Curien was deputy director of the Foundation "Sciences Mathématiques de Paris" until September 2011.

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 9/10 of the issue is already published online).

8.1.3. Program committees and organising committees

Yann Régis-Gianas served on the program committee for the conference "Journees Francophones des Langages Applicatifs".

Yann Régis-Gianas organized a joint workshop regrouping members of the CerCo european project and the Eternal ARC.

Matthieu Sozeau organised a Coq workshop on type-classes and reification in February in Paris, with around 20 attendees. This was funded by the ADT Coq. He was on the organising committee of DTP'11, a satellite workshop of ITP'11 in Nijmegen.

Pierre-Louis Curien coorganised with Paul-André Melliiès the anniversary workshop GGJJ2011, in honour of Gérard Berry and Jean-Jacques Lévy, Gérardmer, February 2011.

Hugo Herbelin was in the Program Committee of the conference TLCA 2011 and of the workshop Coq 2011. He will be in the Program Committee of ITP 2012 and MFPS 2012.

Hugo Herbelin, Alexis Saurin, and Noam Zeilberger organized a satellite workshop of RDP 2011 on "Theory and Practice of Delimited Control" (May).

Hugo Herbelin is organizing the "Automation in Proof Assistant" satellite workshop of ETAPS 2012 which will be partly shared with the organization of the first COST 0901 meeting of year 2012.

Alexis Saurin was in the Program Committee of ICFP 2011 and he organized, together with Marine Meyer, the PC meeting which took place in INRIA offices, gathering the PC members (about 20 PC members) for 3 days in may 2011.

Alexis Saurin was in the Program Committee of the workshop GaLoP 2011.

Alexis Saurin is in the scientific committee of the Logic and Interaction weeks 2011, more specifically in charge of the week on proofs and programs.

8.1.4. Jurys

In June and July 2011, Alexis Saurin has been member of the Jury for entrance exams at Ens Paris and examined the candidates for the three ENS for the "Informatique Fondamentale" Oral exam.

8.1.5. Ph.D. and habilitation juries

Hugo Herbelin (reviewer) and Matthieu Sozeau (member) participated to the PhD committee of François Garillot (École Polytechnique) on December 5th. Hugo Herbelin participated to the PhD committee of Barbara Petit (ENS Lyon) on July 13th.

8.1.5.1. Invited talks

Matthieu Sozeau was invited to the DTP seminar at the Shonan Village Center in Japan, September 2011, and gave a talk on dependent pattern-matching compilation.

Alexis Saurin has been invited to give a talk at a special session on Structural Proof Theory and Computing as part of the 2012 ASL (Association of Symbolic Logic) Annual meeting which will take place in Madison, Wisconsin from March 31-to April 3, 2012.

8.1.5.2. Presentation of papers

Matthias Puech will present his paper [15] at the TLDI 2012 workshop.

Alexis Saurin gave a communication at TLCA 2011 on his work with Zena Ariola and Hugo Herbelin [12]

8.1.5.3. Other presentations

Sozeau and Letouzey presented the developers talk at the Coq'3 workshop in Nijmegen,

Pierre Boutillier, Hugo Herbelin and Noam Zeilberger gave each one a talk at the TPDC workshop in Novi Sad (June).

Arnaud Spiwack and Hugo Herbelin gave a talk at the TYPES conference in Bergen (September).

Matthieu Sozeau and Pierre Letouzey gave a presentation at the Coq workshop 2011 at Nijmegen (August).

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

Guillaume Munch-Maccagnoni gave a talk at the Linear Logic workshop (November, Kyoto).

Noam Zeilberger gave a talk at the European Workshop on Computational Effects in Ljubljana, Slovenia (March).

8.1.5.4. Attendance to conferences, workshops, schools,...

Oregon Programming Languages Summer School (Puech, Sozeau). Journées Francophones des Langages Applicatifs, February, La Bresse (Boutillier).

Final meeting of the ANR project CHOCO (Curry-Howard for concurrency) at Lyon (Munch-Maccagnoni), April.

Hugo Herbelin gave a talk at the FATPA 2011 workshop at Belgrade (February).

Pierre Boutillier and Hugo Herbelin attended TLCA 2011 in Novi Sad.

Hugo Herbelin and Matthieu Sozeau attended ITP 2011 at Nijmegen (August).

Hugo Herbelin attended the Coq workshop 2011 at Nijmegen (August).

Arnaud Spiwack attended CSL 2011.

Matthieu Sozeau and Arnaud Spiwack attended the MAP 2011 conference in Leiden (November).

DTP meeting in Shonan, Japan (Sozeau).

ICFP 2011 in Tokyo (Sozeau).

Hugo Herbelin attended the COST 0901 meeting in Torino.

Noam Zeilberger attended the 13th Agda Implementors' Meeting (AIM XIII) at Chalmers University (Göteborg, Sweden, April).

8.1.5.5. Talks in seminars

Munch-Maccagnoni: "Semantics Lunch", Cambridge Computer Lab (June); Séminaire Théorie des Types et Réalisabilité (PPS, Paris).

Sozeau: Chalmers Programming Languages seminar (Gothenburg, January). IMDEA Seminar (Madrid, October), University of Kent programming languages and systems seminar (Canterbury, November).

Letouzey: talk in a local seminar at University of Nijmegen (April).

Herbelin: talk at the Geocal-Lac meeting of the GDR IM at École Polytechnique (November). Curien and Spiwack attended the meeting.

Herbelin and Sozeau: talks at the Journées PPS, September 2011 (Boutillier, Curien, Glondu, Munch-Maccagnoni, Zeilberger attended the Journées).

Zeilberger: invited speaker at the Paris 7 PhilMaths seminar session on "Proof-theoretic semantics and the justification of logical laws" (March).

Pierre-Louis Curien organised, jointly with Jean-Pierre Jouannaud and Lihong Zhi, a seminar on Rewriting and Algebra at the university of Tsignhua during his three-months visit in China (March-May): he gave two talks, one on Gröbner bases and Rewriting, and another one on Janet bases.

8.1.5.6. 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.

8.1.6. 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 and Guillaume Munch-Maccagnoni 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 gave several conferences about computer science in the high-schools Camille Sée (Paris) and Albert Einstein (Sainte Genevieve des Bois). He also trained four high-school students to be able to explain the first principles of programming to the other students of the high-school.

Yann Régis-Gianas gave short presentations to high-school teachers in order to present the programming part of the new high-school teaching module "Informatique et Science du Numérique" (ISN)

Pierre Letouzey participates in a training period of high-school teachers for the computer science courses that will be introduced next year in the last grade of French high-schools (ISN).

8.2. Teaching

8.2.1. Supervision of Ph.D. and internships

PhD in progress : Pierre Boutillier, Représentation des effets et inférence de type dans le cadre du développement d'un langage de programmation à types riches, September 2010, Hugo Herbelin.

PhD in progress : Stéphane Glondu, Vers une certification de l'extraction de Coq, September 2007, Pierre Letouzey.

PhD in progress : Lourdes del Carmen Gonzalez Huesca, Un langage de tactiques typés pour Coq, December 2011, Hugo Herbelin and Yann-Régis Gianas.

PhD in progress : Guillaume Munch-Maccagnoni, Polarités, réalisabilité classique, contrôle délimité, September 2009, Pierre-Louis Curien and Thomas Ehrhard.

PhD in progress: Pierre-Marie Pédrot,

PhD in progress : Matthias Puech, Incremental proof development, September 2008, Hugo Herbelin, Andrea Asperti, and Yann-Régis Gianas.

Yann Régis-Gianas supervised the master internship of Guillaume Claret.

8.2.2. Courses

Licence : Initiation aux systèmes d'exploitation, (Boutillier) 1/4, L1 sciences exactes, Université Paris Diderot, France

Licence : Algorithmique, (Boutillier) 1/4, L3 BioInfo, Université Paris Diderot, France

Licence : Principes de fonctionnement des machines binaires, (Letouzey) 1/6, L1, Université Paris Diderot, France

Licence : Programmation fonctionnelle, (Letouzey) 1/6, L2 Math-Info, Université Paris Diderot, France

Master : Circuits et architectures, (Letouzey) 1/6, M1, Université Paris Diderot, France

Master : Typage, (Letouzey) 1/6, M2 Pro, Université Paris Diderot, France

Master : Preuves assistées par ordinateur, (Glondu) 1/6, M1, Université Paris Diderot, France

Doctorat : Pierre-Louis Curien and Hugo Herbelin gave lectures at the 10th Oregond Summer School on Programming Languages (June).

Matthias Puech is a teaching assistant (A.T.E.R) at University Paris Diderot – Paris 7 for the academic year 2011–2012. He teaches Java programming and Binary machines (beginner).

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.

9. Bibliography

Major publications by the team in recent years

[1] Z. 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.

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

[3] 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/.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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/.

[10] M. SOZEAU, N. OURY. *First-Class Type Classes*, in "Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings", O. A. MOHAMED, C. MUÑOZ, S. TAHAR (editors), Lecture Notes in Computer Science, Springer, 2008, vol. 5170, p. 278-293.

Publications of the year

Articles in International Peer-Reviewed Journal

[11] V. SILES, H. HERBELIN. *Pure Type System conversion is always typable*, in "Journal of Functional Programming", 2011, to be published, http://hal.inria.fr/inria-00497177/en/.

International Conferences with Proceedings

[12] Z. ARIOLA, H. HERBELIN, A. SAURIN. *Classical Call-by-need and duality*, in "TLCA 2011 - Typed Lambda Calculi and Applications", Novi Sad, Serbia, C.-H. L. ONG (editor), Lecture Notes in Computer Science, Springer, June 2011, vol. 6690, p. 27-44 [*DOI :* 10.1007/978-3-642-21691-6_6], http://hal.inria.fr/inria-00630156/en.

[13] A. SAURIN, M. BASALDELLA, K. TERUI. *On the Meaning of Focalization*, in "workshop on Games, Dialog and Interaction", Paris, France, A. LECOMTE, S. TRONÇON (editors), LNAI, Springer-Verlag, 2011, vol. 6505, p. 78-87, http://hal.inria.fr/hal-00552209/en.

National Conferences with Proceeding

[14] P. BOUTILLIER. *A relaxation of Coq's guard condition*, in "Actes des Journées Francophones des langages Applicatifs", Carnac, France, February 2012, http://hal.archives-ouvertes.fr/hal-00651780/en/.

Conferences without Proceedings

[15] M. PUECH, Y. RÉGIS-GIANAS. *Safe Incremental Type Checking*, in "TLDI 2012 - Seventh ACM SIGPLAN Workshop on Types in Language Design and Implementation", Philadelphia, United States, January 2012, 2 pages, http://hal.inria.fr/hal-00650341/en.

Research Reports

[16] R. M. AMADIO, Y. RÉGIS-GIANAS. *Certifying and reasoning on cost annotations of functional programs*, Inria, October 2011, http://hal.inria.fr/inria-00629473/en.

Other Publications

[17] H. HERBELIN. *A constructive proof of the axiom of dependent choice, compatible with classical logic*, 2011.

[18] H. HERBELIN. *An intuitionistic logic that proves Markov's principle (long version)*, 2011.

References in notes

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

[20] G. BARTHE, J. HATCLIFF, M. H. SØRENSEN. *A notion of classical pure type system*, in "Electr. Notes Theor. Comput. Sci.", 1997, vol. 6, p. 4-59.

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

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

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

[24] T. COQUAND. *Pattern Matching with Dependent Types*, in "Electronic Proceedings of the Third Annual BRA Workshop on Logical Frameworks (Båstad, Sweden)", 1992.

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

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

[31] 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.

[32] H. GOGUEN, C. MCBRIDE, J. MCKINNA. *Eliminating Dependent Pattern Matching*, in "Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday", K. FUTATSUGI, J.-P. JOUANNAUD, J. MESEGUER (editors), Lecture Notes in Computer Science, Springer, 2006, vol. 4060, p. 521-540.

[33] G. GONTHIER, B. ZILIANI, A. NANEVSKI, D. DREYER. *How to make ad hoc proof automation less ad hoc*, in "ICFP", M. M. T. CHAKRAVARTY, Z. HU, O. DANVY (editors), ACM, 2011, p. 163-175.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] D. ILIK. *Delimited control operators prove Double-negation Shift*, 2010, http://hal.inria.fr/hal-00647389/en/.

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

[40] J.-L. KRIVINE. *Structures de réalisabilité, RAM et ultrafiltre sur N*, in "CoRR", 2008, vol. abs/0809.2394.

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

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

[43] 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.

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

[45] 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.

[46] 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.

[47] 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.

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

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