INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team LEMME

Software and mathematics

Sophia Antipolis

THEME SYM

Table of contents

1. Team

Head of project-team
Loïc Pottier [Research scientist INRIA]

Vice-head of project team
Yves Bertot [Research scientist INRIA]

Administrative Assistant
Nathalie Bellesso

Staff members INRIA
Janet Bertot [Research engineer INRIA, service DREAM, at 40 %, until May 2003]
Laurence Rideau [Research scientist INRIA]

Civil servant (on partial secondment)
Philippe Audebaud [Lecturer, ENS Lyon]

Civil servant (on secondment)
Frédérique Guilhot [Qualified teacher, *académie de Nice*]

Post-doctoral fellows
Benjamin Grégoire [Concert]
Hanane Naciri [Mowgli, until September 2004]

Scientific advisors
André Hirschowitz [Professor UNSA, Laboratoire J.A. Dieudonné]
Monica Nesi [Lecturer, University of L'Aquila, Italy]

Ph.D. students
Kuntal Das Barman [Concert, until October 2004]
Assia Mahboubi [Teaching Assistant]

2. Overall Objectives

Formal methods have become increasingly important in software development. The Lemme project aims at contributing to their use in software construction and scientific computing. In particular, we try to bridge the gap between solving a mathematical problem on paper and using a computer, as the latter supports and requires many mechanical computations. Special attention is given to logical consistency and the ease of transfer from theory to practice.

To reach our goal, we work on the following themes:

1. formalization of mathematical theories describing the basic objects of scientific knowledge;
2. development of tools to facilitate the construction of mathematical proofs and efficient implementations, derived from the formal description of an algorithm and its correctness proof; and
3. formalization of programming language semantics;
4. development of the user's working environment in which the certified algorithms and the corresponding proofs are developed. Users can be, for example, researchers, engineers, or students.

Although these themes could have a life of their own, they strongly influence each other. The researcher's working environment needs efficient and correct proof tools. Developing these tools requires the certified description of algorithms, which in turn requires formalizing the basic mathematical tools. Once algorithms are described, it is necessary to implement them in a programming language, whose semantics must be mastered. To complete the circle, formalizing mathematics or programming language semantics can be done more easily and efficiently thanks to a practical working environment.

3. Scientific Foundations

3.1. Type theory and formalization of mathematics

Keywords: *Coq*, *formalization*, *mathematics*, *type theory*.

The calculus of inductive constructions is powerful enough to formalize complex mathematics, based on algebraic structures together with their dependences and operations (as was also done in the Axiom computer algebra system). This is especially important as we want to produce proofs of logical properties for these structures, a goal that is only marginally addressed in most scientific computation systems. The calculus of inductive constructions also makes it possible to write algorithms as recursive functional programs, based on rich data structures. A third important characteristic of the calculus of inductive constructions is that it is also a language for manipulating proofs, thanks to the Curry-Howard isomorphism. All this makes the calculus of inductive constructions a tool of choice for our investigations. However, this language is difficult to learn, like an assembly language, although its conciseness and expressive power make it palatable for experts.

The Coq system and the graphical tool PCoq allow to have nicely readable formalizations: record structures, coercions, PPML pretty-printing, and extensible parsing, for instance, drastically improve readability. But numerous problems are left unsolved: multiple inheritance is not provided; stacking coercions leads to complexity problems; dependent types are awkward to use during proof or object constructions; sub-typing or quotient constructions are lacking.

3.2. Verification of scientific algorithms

Keywords: *Coq*, *algorithms*, *certification*.

To produce certified algorithms, we use the following approach: instead of attempting to prove properties of an existing program (through a formalization of its semantics), we produce programs whose correctness is an immediate consequence of their construction. This has several advantages. First, we work at a high level of abstraction, independently of the target implementation language (but the closer the language to the functional approach, the more efficient the program). Second, we concentrate on specific characteristics of the algorithm, and abstract away from the rest (*e.g.* memory management or data implementation strategies).

However, this approach also presents a few difficulties. For instance, it is still difficult to prove properties of recursive algorithms where termination is ensured thanks to a well-founded order. It is also difficult to work in an imperative style or with operations that have side-effects.

3.3. Programming language semantics

Keywords: *Coq*, *programming languages*, *semantics*.

We also investigate the algorithms that occur when implementing programming languages. For these algorithms, we generally base our work on the semantic description of a language. The properties that we attempt to prove for an algorithm are, for example, that it preserves the semantics of programs (when the algorithm is a transformation or optimization algorithm) or that the programs produced are free of some unwanted behavior (when the algorithm is a compiler or a program verifier). For these algorithms, the complexity sometimes lies in the size of the language description, and sometimes in the intrinsic complexity of the algorithm, which may use other algorithms such as graph traversal or unification algorithms. We usually talk about "proofs in programming language semantics" to refer to this class of algorithms and their correctness proofs. In addition to its intrinsic interest, this work is also useful for our work on scientific algorithms. Using the semantics of the programming language used to implement the algorithms, we can guarantee the correctness and efficiency of these implementations.

3.4. Proof environments

Keywords: *Coq*, *environments*, *man-machine interface*, *proofs*.

We study how to improve mechanical tools for searching and verifying mathematical proofs used by engineers and mathematicians to develop software and formal mathematical theories. There are two complementary objectives. The first is to improve the means of interaction between users and computers, so that the tools become usable by engineers, who have otherwise little interest in proof theory, and by mathematicians, who have little interest in programming or other kinds of formal constraints. The second objective is to make it easier to maintain large formal mathematical developments, so they can be re-used in a wide variety of contexts. Thus, we hope to increase the use of formal methods in software development, both by making it easier for beginners and by making it more efficient for expert users.

4. Application Domains

4.1. Certified scientific algorithms

For some applications, it is mandatory to build zero-default software. One way to reach this high level of reliability is to develop not only the program, but also a formal proof of its correctness. In the Lemme team, we are interested in certifying algorithms for scientific computing. For this, we propose a methodology that consists in starting not with the program but with the abstract algorithm. Proving first the algorithm's correctness has the main advantage that the proof usually contains all the deep mathematical properties. The second step that consists in deriving an efficient implementation from an algorithm could then be handled automatically or semi-automatically. We have already concluded several experiments in the area of computer arithmetic, polynomial computations, and computational geometry.

4.2. Web, MathML, XML

Our work around XML and MathML, within the context of Figue, has two important goals:

- it should enable us to share proofs (represented as Coq scripts) on the web, by generating XML + MathML representations of proofs from our PCoq interface. These XML+MathML proof representations can then be displayed (and printed) with “real" mathematical formulae by any navigator supporting MathML, thus making the proof representation independent of PCoq.
- continuing the work of Loïc Pottier on Wims (WWW Interactive Mathematics Server, developed by Xiao Gang at the University of Nice), we experiment with building proofs directly on the web, with 10 teachers and 200 students.

In the long run, the shift of attention towards the web should increase the visibility of our work, so that we can have a larger group of users for our tools. The European contract LTR Mowgli will significantly support this.

5. Software

5.1. PCoq

Participants: Janet Bertot, Yves Bertot [correspondent], Loïc Pottier, Laurence Rideau.

We distribute the PCoq system, a front-end to the Coq theorem prover, which allows a better handling of mathematical notations. It has been used outside the team for several large scale proof developments.

5.2. Aïoli and Figue

Participants: Hanane Naciri, Laurence Rideau [correspondent], Laurent Théry.

Aïoli and Figue are base components of PCoq. In PCoq, mathematical formulae are represented as trees. Aïoli defines the manipulation of these trees, while Figue defines appropriate algorithms for displaying.

Particular attention is given to two-dimensional displaying of matrices, fractions, square roots, *etc.* For more information, see http://www-sop.inria.fr/lemme/aioli/doc/aioli.html and http://www-sop.inria.fr/lemme/figue/.

6. New Results

6.1. Tools for proof environments

6.1.1. Pcoq

Participants: Janet Bertot, Yves Bertot [correspondent], Loïc Pottier, Laurence Rideau.

There were two tasks on Pcoq this year: the first one was to adapt it to the new version of Coq (which included a drastic change in syntax); the second was to refactor the history management and classify commands according the kind of effect they produce to make Pcoq more independentant of the undelying Coq syntax.

6.1.2. Latex to HTML converter

Participants: Hanane Naciri, Loïc Pottier.

We make scientific documents written in Latex accessible on the Web, while handling their structured objects like formulae. We first convert the Latex to XHTML+MathML, then convert it to HTML+images. For each formula, our program produces an image with specific sub-areas and associates each area with the corresponding MathML structure (a sub formula) that can be manipulated. This gives mathematicians a simple way to obtain an active web version of their Latex documents, even if it is maybe not the most efficient way.

6.1.3. Proof explanations: using natural language and graph views

Participants: Frédérique Guilhot, Hanane Naciri, Loïc Pottier.

The aim of this work is to generate automatically from a Coq proof script (a set of commands given to Coq to perform the proof) an explanatory text in natural language (in French or in English) and a deduction graph. Hence formal proofs can be understood by people who are not familiar with proof assistants. To provide these views, we first translate the proof script into an XML tree. Then we visualize this tree as a structured text of explanations written using forward style (from assumptions to conclusion). These explanations are presented in a Web document with appropriate mathematical notations.

The XML tree can also be visualized using a deduction graph, an acyclic oriented graph in which each reasoning step is represented by a subgraph. In this view, only facts and the links between them are represented. This is a non linear presentation that gives a global view of the proof.

6.1.4. Mowgli prototype

Participants: Yves Bertot, Hanane Naciri, Laurence Rideau.

We have improved the Mowgli prototype. This prototype is a web interface giving access to the Mowgli library (which contains proof data from Coq and scientific papers in Tex source). This prototype gives proof developers the possibility to contribute to the Mowgli library. Proof developers can store their proofs in the Mowgli library, and browse through their own or other proofs. The Mowgli prototype gives both the content view (lambda terms, and proof trees in XML) and the natural language explanation view of the proof data (presented in HTML or XHTML+MathML). For more information, see http://www-sop.inria.fr/lemme/Hanane.Naciri/Mowgli/.

6.1.5. Using Coq on the web

Participant: Loïc Pottier.

With 10 teachers in mathematics at Nice and 200 students at the first level of University in mathematics, we experiment a tool allowing to do proofs interactively, in coq, via a browser and a web serveur. This experience began in october 2004, and consists in about 3 exercices per week in analysis.

6.1.6. Geoview

Participants: Frédérique Guilhot, Loïc Pottier.

Geoview is a tool that allows to make a drawing from a theorem in planar geometry. It uses a Java applet (developed by F. Koteki of CNAM) to show the drawing on the screen, and is integrated into the PCoq interface. This work is described in the article [4] that is now accepted for publication.

6.1.7. TeXMacs

Participants: Philippe Audebaud, Laurence Rideau.

An article describing our work on combining TeXMacs and Coq for the publication of formal proofs has been published [3].

6.2. Type theory and formalization of mathematics

6.2.1. Type theory

Participants: Yves Bertot, Pierre Castéran [U. Bordeaux/INRIA Futurs].

The book by Yves Bertot and Pierre Castéran on the theoretical and practical aspects of the Coq System is now published [1].

6.2.2. Toward Geometric Views on the Λ-Calculus

Participant: Philippe Audebaud.

In this study, our initial language is the pure λ-calculus. We take the view that types are properties for terms, which leads us to types assignment systems (also known as intersection types).

Whichever view of types we have in mind, it is commonly admitted since Tarski and Stone that they should be interpreted as open sets of some topogical space. However, the property which is required for their various interpretations (normalisation, realisability, full abstraction) is stability by finite and arbitrary intersections of subsets, which rather advocate for interpretation as closed sets.

In our interpretation, types are closed sets of λ-terms which fulfil some particular property (strong normalisation, confluence, ...). We are still far from a complete answer because pure λ-calculus does not provide any native operation on terms which allows finite unions of (types as) closed sets to be the collection of terms which share some property. We show that adding the parallel operator introduced by Boudol is a key ingredient for building an interpretation of the resulting typed calculus using the closed sets of a particular class of topologies.

Further analysing the problem, we show that this parallel operator is actually the multiplication for a ring structure which is adjoined to the initial λ-calculus. Our extended calculus, named $\mathbb{E}$ is thus the combination of both a structure of commutative integral ring and λ-calculus; it extends pure λ-calculus along the same lines as relative numbers $\mathbb{Z}$ extend natural numbers $\mathbb{N}$, the ring operations expressing computation rules on terms.

In this new setting, types are naturally interpreted as closed sets of some Zariski topology on $\mathbb{E}$: zeroes sets for some notion of polynomial ideals (algebraic sets). Term properties (strong normalisation, confluence, full abstraction) are investigated along the line of a single generic interpretation.

This first experiment suggests a lot of further developments based on algebraic geometry. For instance, converging programs appear to be in a hyperplan, which suggests that they are rare.

6.2.3. Partial Co-recursion

Participant: Yves Bertot.

Recursive functions can be designed to produce conceptually infinite data structure and consume these data-structures on a call-by-need basis. Previous work by T. Coquand, L. Paulson, and E. Gimenez showed that these functions belonged to a class of recursive functions that is different from the class that is usually considered in theorem provers. This new class of functions is called the class of co-recursive functions and

the infinite data-structures are known as co-inductive structures. However, none of the previous studies had considered partial functions.

We have described a solution to consider partial functions, with a special application on filters for infinite streams. An example was given on Eratosthene's sieve on infinite streams of numbers for which a complete formal proof of correctness has been worked out. A paper describing this work is submitted for publication [8]. For the future, we expect our solution to be implemented in an automatic processor. Applications of this technique can also be interesting for exact real number computations [7].

6.2.4. Recursive functions

Participant: Yves Bertot.

We have developed a collection of algorithms to compute square, cubic, and nth root of numbers in binary form. This example can be used as teaching material for the tools we have designed for the definition of well-founded recursive functions [9]. This work has been submitted for publication and is currently being revised.

6.2.5. Termination

Participants: Yves Bertot, Bejamin Grégoire, Gilles Barthe [projet Everest], Fernando Patawski [projet Everest].

Different techniques to ensure the termination of dependently typed functions have been studied in the team this year.

First a technique relying on staged types, where the types carry information about the respective size of input and output of functions has been explored and a prototype has been implemented for testing in the framework of the calculus of inductive constructions. An article on this topic has been submitted for publication.

Second a method that takes advantage of the current capabilities of the guard systems implemented in the Coq system has been designed. Our experiments show that the implemented guard system is more powerful than its published versions. We are collaborating with researchers from the LOGICAL team to publish a better description of the guard system.

6.2.6. Complexity issues

Participant: Yves Bertot.

For the need of theorem proving, pattern-matching constructs are compiled to constructs where the matching rules are elementary and pairwise exclusive. This expansion gives rise to exponential increases in code size and the proving tools then face a complexity wall. We have shown that this combinatorial explosion could be avoided by using a method that keeps the "order" of matching rules and confines the combinatorial explosion in one single theorem.

6.2.7. Formalization in Coq of high-school mathematics

Participants: Frédérique Guilhot, Loïc Pottier.

Frédérique Guilhot has developed and improved a library dedicated to high-school geometry for Coq. An article describing this work has been published [6].

A new topic, concerning high-school level probability calculus has been started.

6.2.8. A sharp efficiency increase for ring equalities

Participants: Benjamin Grégoire, Assia Mahboubi, Loïc Pottier.

We have explored a new approach to proving equalities between polynomial expressions in a ring using reflection. This new approach is based on the Hörner encoding of polynomial expressions. Based on a variety of test cases, this new approach proves to be much more efficient than the traditional approach based on full development and ordering of monomials. We plan to perform a formal study of this method's complexity and compare it with the traditional approach.

6.2.9. Compiled reduction for the calculus of constructions

Participant: Benjamin Grégoire.

One of the needs of reduction in dependently typed calculi like the calculus of constructions is to perform comparison of terms during type-checking. This use of reduction can be particularly computation-intensive in the context of proofs by reflection. An approach to make reduction efficient is to compile terms of the calculus of construction into a byte code similar to the one used for Ocaml, and to run this compiled program. We designed a solution along this line during the Doctorate's work performed in the CRISTAL and LOGICAL projects. This year, we integrated this approach inside the main development version of the Coq system.

Experimental results fall in three categories:

1. Proofs that rely on reflection benefit from a sharp increase in efficiency,
2. Most of the other proofs have their verification time practically unchanged,
3. A few examples suffer from a drastic deterioration of efficiency. These cases still need to be investigated.

6.2.10. Proving polynomial inequalities with real coefficients in Coq

Participants: Assia Mahboubi, Loïc Pottier.

Assia Mahboubi began to implement in Coq the cylindric algebraic decomposition of Collins. The next step wil be to prove this implementation. The goal is to have a tactic solving efficiently polynomial inequalities with real coefficients in Coq.

6.3. Programming language semantics

6.3.1. A Cminor to RTL translator

Participants: Yves Bertot, Kuntal Das Barman, Xavier Leroy [project-team CRISTAL].

We modeled the translator from Cminor to an intermediate language as a collection of mutually recursive and side-effect free functions, which respect the definition restriction provided in the theorem prover Coq. We then addressed the question of proving that this part of the compiler is correct. While working on a first partial correctness result (the result graph is well-connected: all successors of all nodes are also present in the graph) we observed that the proof complexity was too large to be handled with usual techniques. The source of complexity is that the translator performs a few optimisations on the fly, using deep patterns to recognize the optimization opportunities and these patterns are not handled well by the theorem prover. We devised a proof method that makes it possible to work around this difficulty and completed the proof for the partial correctness statement. We expect the new method will also make the complete correctness statement amenable.

An alternative workaround to the complexity problem has also been studied. This alternative workaround is to re-implement the compiler to avoid the deep patterns and prove the optimization patterns separately. This method has proved efficient and the complete proof could be be performed on a re-designed part of the compiler.

6.3.2. Optimizations at the RTL-CFG level

Participant: Benjamin Grégoire.

We studied some of the optimizations usually performed at the level of the intermediate language (a Register-Transfer-Language where the Control Flow Graph appears immediately) in a compiler. We rephrased these optimisations as instances of a general framework using first an analysis phase and second a transfer phase. For each optimization, the analysis uses Kildall's algorithm to find the fixpoint of a monotonic function in a lattice. An important part of our work was to exhibit the well-founded relations for different lattices, as required by Kildall's algorithm. We expect this kind of work to become easier and easier, thanks to better and better libraries of results on well-foundedness that we develop.

6.3.3. Parallel Move

Participants: Laurence Rideau, Bernard Serpette [project-team OASIS].

We studied a well-known algorithm for the parallel assignment of registers, where the values of the target registers are taken from a collection of source registers, among which the target registers may also occur. The difficulty is to find a suitable order so that the value of a source register is not overwritten before it has been moved to a target register, while using only one extra temporary register. This algorithm has been formally proved and executable code for integration in the proved compiler has been generated from the proof. An article describing this work has been submitted for publication.

6.3.4. Linearization

Participants: Benjamin Grégoire, Xavier Leroy [project-team CRISTAL].

We have designed and proved correct an algorithm that maps a graph of RTL instructions to a sequence of instructions, closer to the usual form of assembly language programs. This is where `goto` statements are introduced. Our algorithm takes care of minimizing the number of branches.

7. Contracts and Grants with Industry

7.1. Mowgli

We participate in the European LTR project Mowgli. Other participants are the Universities of Bologna (Italy), Berlin (Germany), Nijmegen (Netherlands) and Eindhoven (Netherlands), the DFKI (Saarbrücken, Germany), the Max Planck Institute (Germany), and Trusted Logic. The goal of the project is to build on top of previous standards for the management and publishing of mathematical documents (MathML, OpenMath, OMDoc), and to integrate them with different XML technologies (XSLT, RDF, *etc.*).

8. Other Grants and Activities

8.1. International collaborations

- ORCCA - Ontario Research Center for Computer Algebra, Canada : collaboration around MathML (bi-directional displaying of mathematical formulae).
- Universities of Bologna (Italy) and Nijmegen (Netherlands), DFKI (Saarbrücken, Germany) and the Max Planck Institute for Gravitational Physics (Germany): mathematics on the Web.

8.2. National initiatives

- INRIA New Investigation grant (ARC) Concert (*Compilateur Certifié*, Certified compiler). Other participants are Cristal (Rocquencourt), Miró (Sophia), Oasis (Sophia), Mimosa (Sophia), and IIE-CNAM (Evry). Several meetings have been held during the year, and the project shares a CVS repository and a mailing-list to gather data around the development of a certified compiler for a subset of C called Cminor. For more information, see http://www-sop.inria.fr/lemme/concert/.

8.3. European initiatives

Lemme participates in the networks Types (type theory).

9. Dissemination

9.1. Conference and workshop attendance, travel

Yves Bertot Yves Bertot attended a workshop in Dagstuhl in September on dependently types programming where he presented his experiment on Eratosthene's sieve. He was invited to a workshop on Co-induction in Dresden in September,where he presented the capabilities of co-inductive tools in the Coq system. He gave talks on partial co-recursive functions at a workshop of the ACI GéoCal in Marseille in November and at the workshop of the Types european coordination action in Jouy-en-Josas in December.

Frédérique Guilhot Frédérique Guilhot attended the conference JFLA'04 in January.

Loïc Pottier Loïc Pottier presented a paper on a new tactic proving equalities with Groebner bases in Coq at JFLA 2004 in january. He presented his work on interactive proofs to some mathematicians in Marseille in June, and was invited to gave a talk on this subject at a "Small Types Workshop" in Nijmegen in october. He attended the "Assises nationale de la recherche" in Grenoble in october, as a delegate (?) of the organisation comitee of Nice.

Laurence Rideau Laurence Rideau attended the conference JFLA'04 in January.

9.2. Leadership within scientific community

- Yves Bertot was a member of the program committee for TPHOLS'04.
- Loïc Pottier is a member of the program committee for JFLA 2005.
- Project members reviewed papers for among others TSI, RNC6, ENTCS, RTA.

9.3. Miscellaneous

- Yves Bertot is a member of the jury for the SPECIF thesis award.
- Yves Bertot is a member of the *Conseil National des Universités* (National University Council), 27th section. This position brings more than a month of work in reviewing applicant profiles for university professor positions.

9.4. Supervision of Ph.D. projects

- Yves Bertot supervises the Ph.D. projects of Kuntal Das Barman (completed October 2003).
- Loïc Pottier supervises the Ph.D. projects of Assia Mahboubi and Frédérique Guilhot.

9.5. Teaching

Philippe Audebaud *Compilation, analyse syntaxique* 26 heures Licence, *Compilation, code generation*, Maitrise 26 heures *Sémantique des langages de programmation*, Maitrise, 18 heures, *Algorithmique théorique* Deug2, *48h*, plus divers TD d'algorithmique et de langages (Scheme), pour un total de 200 heures.

Yves Bertot *Sémantique des langages de programmation I* (Programming language semantics I), Maîtrise Informatique (4th year, 14 hours), University of Nice.
Programmation fonctionnelle et preuves (Proofs and Functional Programming), Maîtrise, (4th year, 32 hours), ENS Lyon, *Sémantique des langages de programmation II* (Programming language semantics II), 2nd year Master (5th year, 24 hours), University of Nice, *Proof mecanisation*, 2nd year Master (5th year, 6 hours).

Loïc Pottier *Preuves formelles: théorie et applications* (Formal Proofs: theory and applications), DEA mathematics, University of Nice.
Logique et informatique (Logic and Computer Science), Maîtrise MIM, University of Nice.

10. Bibliography

Books and Monographs

[1] Y. BERTOT, P. CASTÉRAN. *Interactive Theorem Proving and Program development*, Springer, 2004.

Doctoral dissertations and Habilitation theses

[2] K. D. BARMAN. *Type theoretic semantics for programming languages*, Ph. D. Thesis, Université de Nice-Sophia-Antipolis, September 2004.

Articles in referred journals and book chapters

[3] P. AUDEBAUD, L. RIDEAU. *TeX as Authoring Tool for Formal Developments*, in "Electronic Notes in Theoretical Computer Science", vol. 103, 2004.

[4] Y. BERTOT, F. GUILHOT, L. POTTIER. *Visualizing Geometrical Statements with GeoView*, in "Electronic Notes in Theoretical Computer Science", vol. 103, 2004.

Publications in Conferences and Workshops

[5] J. CRECI, L. POTTIER. *La tactique GB*, in "JFLA 2004", 2004.

[6] F. GUILHOT. *Formalisation en Coq d'un cours de géométrie pour le lycée*, in "JFLA 2004", 2004.

[7] M. NIQUI, Y. BERTOT. *Qarith: Coq Formalisation of Lazy Rational Arithmetic*, Lecture Notes in Computer Science, vol. 3085, Springer-Verlag, 2004, p. 309–323.

Internal Reports

[8] Y. BERTOT. *Filters on Co-Inductive streams: an application to Eratosthene's sieve*, Technical report, n° RR-5343, INRIA, 2004, http://www.inria.fr/rrrt/rr-5343.html.

[9] Y. BERTOT. *Vérification formelle d'extractions de racines*, En Français, Technical report, n° RR-5344, INRIA, 2004, http://www.inria.fr/rrrt/rr-5344.html.