INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team Cristal

Type-safe programming, modularity and compilation

Rocquencourt

THEME SYM

Table of contents

1. Team

Head of project-team
Xavier Leroy [Senior research scientist (DR), INRIA]

Vice-head of project-team
Didier Rémy [Senior research scientist (DR), INRIA]

Administrative assistant
Nelly Maloisel [TR INRIA]

INRIA staff
Damien Doligez [Research scientist (CR)]
Alain Frisch [Research scientist (Corps des Télécoms), since July 2004]
Maxence Guesdon [Technical staff (IR), 20% Cristal, 80% Miriad]
Gérard Huet [Senior research scientist (DR), also affiliated with project Signes]
Michel Mauny [Senior research scientist (DR)]
François Pottier [Research scientist (CR)]
Pierre Weis [Senior research scientist (DR)]

Visiting staff
Sandrine Blazy [Assistant professor, CNAM/IIE, since September 2004]
Basile Starynkevitch [Senior research staff, CEA, until September 2004]
Hanfei Wang [Associate professor, Wuhan University, until October 2004]

Ph.D. students
Richard Bonichon [MENRT grant, university Paris 6]
Daniel Bonniot [university Paris 7]
Nadji Gauthier [MENRT grant, university Paris 7]
Yann Régis-Gianas [MENRT grant, university Paris 7, since October 2004]
Vincent Simonet [AMN, university Paris 7, until April 2004]
Boris Yakobowski [ENS Cachan, university Paris 7, since October 2004]

Student interns
David Haguenauer [University Paris 6, April–September 2004]
Grégoire Henry [University Paris 6, April–July 2004]
Gaëlle Le Page [EPITA, October–December 2004]
Yann Régis-Gianas [University Paris 7, April–September 2004]
Burhan Ulhaq [University Paris 6, May–August 2004]
Boris Yakobowski [ENS Cachan, April–September 2004]

2. Overall Objectives

The research carried out in the Cristal group is centered on type systems and related program analyses, applied to functional, object-oriented, and modular programming languages. The Caml language embodies many of our research results. Our work spans the whole spectrum from theoretical foundations and formal semantics to language design, efficient and robust implementations, and applications to real-world problems. The conviction of the Cristal group is that high-level, statically-typed programming languages greatly enhance program reliability, security and efficiency, during both development and maintenance.

3. Scientific Foundations

3.1. Type systems

Typing is a fundamental concept in programming: it allows the specification and automatic verification of consistent handling of data in programs. Moreover, when applied to functions, classes, and modules, typing helps structuring and managing large and complex programs.

The Cristal group studies type systems and typing techniques with the aim of designing safe programming languages and tools, whose properties are formally established.

3.1.1. The Hindley-Milner type system

The ML language, designed circa 1978, supports automated compile-time type inference and checking. Its type system supports a form of *parametric polymorphism* that allows giving types to generic algorithms, that is, algorithms that work uniformly on a variety of data types. This type system, as well as its type inference algorithm, are formally specified. Its main property is expressed through a soundness theorem: *Well-typed programs cannot go wrong*.

The ML type system has been the subject of numerous studies and extensions during the last two decades. Even if they are meant to be reusable in other contexts, many of the results of the Cristal group in this area are primarily applied in the Caml language, a dialect of ML developed in the Cristal group.

3.1.2. Typing objects

Although object-oriented programming is widely used in industry, very few OO programming languages have a clear semantics and a formal type system. Different approaches have been proposed in order to lay firm type-theoretic foundations for OO programming.

The main approach followed by the Cristal group relies on Didier Rémy's PhD work [46] that proposed a type system for extensible records allowing type inference. This approach was first extended to objects by Rémy, in his ML-ART prototype [47]. Rémy and Vouillon [48] later added a fully-fledged class-based object layer to Caml, which then became Objective Caml (OCaml). This OO layer is compatible with type inference, and supports advanced OO concepts, such as multiple inheritance, parameterized classes, and "my type" specialization. It can statically type idioms, such as container classes or binary methods, that require dynamic type-checking in conventional OO languages, such as Java.

Another approach to type inference for object-oriented programming, featuring implicit subsumption and based on subtyping constraints, was studied by François Pottier in his PhD dissertation [6][7]. Our current efforts are directed towards multi-methods (that is, method dispatching based on all the arguments of a method) and the introduction of objects in concurrent programming languages, such as the ones based on the join-calculus [41].

3.1.3. Typing modules

Module systems provide an alternative to classes for structuring and decomposing large programs. The Caml module system [44] provides advanced mechanisms for packaging data types with associated operations, maintaining multiple, possibly abstract interfaces for these packages, and parameterizing a module over other modules. It is based on the Standard ML module system but offers improved support for separate compilation, through strict adherence to purely syntactic module types.

While modules are fundamentally different from objects and classes, it would be preferable for modules to take advantage of the possibilities offered by classes such as inheritance and overriding, as well as mutual recursion. These issues are currently studied by the Cristal group in the general framework of *mixin modules* [39].

Finally, extensional polymorphism provides identifier overloading and generic functions, whose behavior is guided by the type of their arguments. Extensional polymorphism [42] also allows modelling computations that dynamically depend on types (input/output, values with dynamic types).

3.1.4. First-class polymorphism and higher-order types

On the one hand, the Hindley-Milner type system has a restricted form of polymorphism: its types are first-order (they do not contain universal quantifiers), and universal quantification is allowed only at the top level, as part of type schemes. This restriction allows ML to support automated type inference, without the need for type annotations in programs.

On the other hand, system F allows universal quantifiers to occur arbitrarily deep within a type expression. This "first-class" polymorphism provides greater expressiveness, at the expense of type inference, which becomes undecidable.

System F's types are second order, i.e., universal quantification is limited to types. In addition, system F_ω offers higher-order types: it also allows universal quantification over type functions (or, in other words, over parameterized types). This provides even greater expressiveness, making it possible to express notions in the core language, which in ML must be relegated to the level of the module language.

The search for type systems that combine some form of first-class and possibly higher-order polymorphism with a decidable or tractable type inference problem, constitute an important research objective at Cristal [2][4]. Results in this area would increase the expressiveness of ML-style languages, strengthen their abstraction capabilities, and therefore facilitate code reuse.

3.1.5. Typing and confidentiality

Numerous programs manipulate sensitive corporate or private data. A static *information flow* analysis provides a way of preventing such confidential information from being leaked to untrusted third parties. The Cristal group applies type inference techniques to this problem [8].

3.2. The Caml programming language

Caml belongs to the ML family of languages. Like all languages of this family, Caml supports both functional and imperative programming styles, Hindley-Milner static typing with type inference, and features a powerful module system. Caml also supports class-based object-oriented programming, and has an efficient compiler.

Initially built around a functional kernel with imperative extensions, the ML language evolved in two independent ways during the 80's. First, a group headed by Robin Milner designed Standard ML (SML), whose most innovative feature was its module system, designed by David MacQueen. In parallel, Caml was first designed and developed in the Formel group at INRIA, in collaboration with members of the C.S. Lab. of École Normale Supérieure in Paris, and then in the Cristal group. Xavier Leroy designed and implemented a module system similar to that of SML, but with a greater focus on separate compilation. Didier Rémy and Jérôme Vouillon built the object layer. The version of Caml including these features is called Objective Caml (OCaml).

From an implementation point of view, Caml has advanced the state of the art in compiling functional languages. Initially based on the Categorical Abstract Machine, which was implemented on top of a Lisp runtime system, Caml's execution model was first changed to a high-performance bytecoded virtual machine designed by Xavier Leroy and complemented by Damien Doligez's generational and incremental garbage collector [40]. The resulting implementation, Caml-Light, has low memory requirements and high portability, which made it quite popular for education.

Then, on the way from Caml-Light to Objective Caml, Xavier Leroy complemented the bytecode generator with a high-performance native code compiler featuring optimizations based on control-flow analyses, good register allocation, and code generators for 9 different processors. The combination of the two compilers provides portability and short development cycle, thanks to the bytecode compiler, as well as excellent performance, thanks to the native code generator.

4. Application Domains

4.1. Software reliability

One of the aims of static typing is early detection of programming errors. In addition, typed programming languages encourage programmers to structure their code in ways that facilitate debugging and maintenance. Judicious uses of type abstraction and other encapsulation mechanisms allow static type checking to enforce program invariants.

Typed functional programming is also an excellent match for the application of formal methods: functional programs lend themselves very well to program proof, and the Coq proof assistant can extract Caml code directly from Coq specifications and proofs.

4.2. Processing of complex structured data

Like most functional languages, Caml is very well suited to expressing processing and transformations of complex, structured data. It provides concise, high-level declarations for data structures; a very expressive pattern-matching mechanism to destructure data; and compile-time exhaustiveness tests. (Languages such as XDuce and CDuce extend these benefits to the handling of semi-structured XML data.) Therefore, Caml is a suitable match for applications involving significant amounts of symbolic processing: compilers, program analyzers and theorem provers, but also (and less obviously) distributed collaborative applications, advanced Web applications, financial analysis tools, etc.

4.3. Fast development

Static typing is often criticized as being verbose (due to the additional type declarations required) and inflexible (due to, for instance, class hierarchies that must be fixed in advance). Its combination with type inference, as in the Caml language, substantially diminishes the importance of these problems: type inference allows programs to be initially written with few or no type declarations; moreover, the OCaml approach to object-oriented programming completely separates the class inheritance hierarchy from the type compatibility relation. Therefore, the Caml language is perfectly suitable for fast prototyping and the gradual evolution of software prototypes into final applications, as advocated by the popular "extreme programming" methodology.

4.4. Programming secure applications

Strongly-typed programming languages are inherently well-suited to the development of high-security applications, because by design they prevent a number of popular attacks (buffer overflows, executing network data as if it were code, etc). Moreover, the methods used in designing type systems and establishing their properties are also applicable to the specification and verification of security policies.

4.5. Web programming

Web programming is an application domain on which the Cristal group has worked in the past, e.g. via the MMM applet-enabled browser [45] and the V6 smart Web proxy. Currently, the use of XML documents, online or offline, and the need for typing their transformations opens an application area that perfectly matches the strengths of languages such as OCaml. Our recent work in this area concentrates on statically-typed XML transformation languages such as XDuce and CDuce, where XML DTDs and Schemas are interpreted as types and statically enforced through type-checking of XML transformations. We also study the integration of these domain-specific languages for XML processing with general-purposes languages such as OCaml.

4.6. Teaching programming

Our work on the Caml language has an impact on the teaching of programming. Caml Light is one of the programming languages selected by the French Ministry of Education for teaching Computer Science in French *classes préparatoires scientifiques*. Caml Light and OCaml are also used in engineering schools, colleges and universities in France, the US, and Japan.

4.7. Computational linguistics

Computational linguistics focuses on the processing of natural languages by computer programs. This rapidly expanding field is multi-disciplinary in nature, and involves several areas that are extensively represented at INRIA: syntactic analysis, computational logic, type theory. During the last two years, Gérard

Huet has been investigating this new domain, and has shown that OCaml and its Camlp4 preprocessor have been highly effective for the development of natural language processing components.

5. Software

5.1. Advanced software

The following software developments are publicly distributed (generally under Open Source licenses), actively supported, and used outside our group.

5.1.1. Caml Light

Participants: Xavier Leroy, Damien Doligez, Pierre Weis.

Caml Light is a lightweight, portable implementation of the core Caml language. It is still actively used in education, but most other users have switched over to its successor, Objective Caml.

Web site: http://caml.inria.fr/.

5.1.2. Objective Caml

Participants: Xavier Leroy, Damien Doligez, Jacques Garrigue [Kyoto University], Maxence Guesdon, Luc Maranget [project Moscova], Jérôme Vouillon [CNRS, U. Paris 7], Pierre Weis.

Objective Caml is our main implementation of the Caml language. From a language standpoint, it extends the core Caml language with a fully-fledged object and class layer, as well as a powerful module system, all joined together by a sound, polymorphic type system featuring type inference. The Objective Caml system is an industrial-strength implementation of this language, featuring a high-performance native-code compiler for 9 processor architectures (IA32, PowerPC, AMD64, Alpha, Sparc, Mips, IA64, HPPA, StrongArm), as well as a bytecode compiler and interactive loop for quick development and portability. The Objective Caml distribution includes a comprehensive standard library, as well as a replay debugger, lexer and parser generators, and a documentation generator.

Web site: http://caml.inria.fr/.

5.1.3. Camlp4

Participant: Michel Mauny.

Camlp4 is a source pre-processor for Objective Caml that enables defining extensions to the Caml syntax (such as syntax macros and embedded languages), redefining the Caml syntax, pretty-printing Caml programs, and programming recursive-descent, dynamically-extensible parsers. For instance, the syntax of OCaml streams and recursive descent parsers is defined as a Camlp4 syntax extension. Camlp4 communicates with the OCaml compilers via pre-parsed abstract syntax trees. Originally developed by Daniel de Rauglaudre, Camlp4 has been maintained since 2003 by Michel Mauny.

Web site: http://caml.inria.fr/camlp4/.

5.1.4. CDuce

Participants: Alain Frisch, Giuseppe Castagna [ENS Paris], Véronique Benzaken [LRI, U. Paris Sud].

CDuce is a functional programming language adapted to the safe and efficient manipulation of XML documents. Its type system ensures the validity (with respect to some DTD) of documents resulting from a transformation on valid inputs. CDuce features higher-order and overloaded functions, implicit subtyping, a powerful pattern matching operations based on regular expression patterns, and other general purpose constructions.

Starting from the 0.2.0 release, CDuce includes a typeful interface with Objective Caml, which allows a smooth integratation of the two languages in a complex project. CDuce units can use existing Objective Caml libraries without the burden of writing any stub code, and they can themselves be used from Objective Caml.

Web site: http://www.cduce.org/.

5.1.5. Cameleon

Participants: Maxence Guesdon, Pierre-Yves Strub [ingénieur expert MIRIAD].

Cameleon is a customizable integrated development environment for Objective Caml, providing a smooth integration between the Objective Caml compilers, its documentation, standard editors, a configuration management system based on CVS, and specialized code generation tools, such as DBForge (stub generator for accessing SQL databases).

Web site: http://savannah.nongnu.org/projects/cameleon.

5.1.6. ActiveDVI

Participants: Pierre Weis, Didier Rémy, Roberto Di Cosmo [U. Paris 7], Jun Furuse [U. Tokyo], Alexandre Miquel [U. Paris 7].

ActiveDVI is a programmable viewer for DVI files produced by the TeX and LaTeX text processors. It provides many fancy graphic effects and can use any X Windows application as plug-in, therefore allowing a demo to be embedded in a presentation, for instance. ActiveDVI provides an excellent Unix/LaTeX-based alternative to PowerPoint presentations. In particular, it supports time recording in slide shows: a lecture can be post-synchronized with the speaker's words. ActiveDVI uses the Camlimages library developed by J. Furuse and P. Weis for displaying embedded images.

Web site: http://pauillac.inria.fr/advi/.

5.1.7. SpamOracle

Participant: Xavier Leroy.

SpamOracle is an automatic e-mail classification tool that separates legitimate e-mail from unsolicited commercial e-mail (spam). It proceeds by Bayesian statistical analysis of word frequencies, based on a user-provided corpus of legitimate and spam messages.

Web site: http://cristal.inria.fr/~xleroy/software.html.

5.1.8. WhizzyTeX

Participant: Didier Rémy.

WhizzyTeX is an Emacs extension that allows previewing of a LaTeX document in real-time during editing.

Web site: http://pauillac.inria.fr/whizzytex/.

5.2. Prototype software

The following software developments are prototypes used mostly within our group, either for experimental purposes or to support our specific needs. Nonetheless, they are all publicly distributed.

5.2.1. Flow Caml

Participant: Vincent Simonet.

Flow Caml is a prototype implementation of an information flow analyzer for the Caml language. It extends Objective Caml with a type system tracing information flow. Its purpose is basically to enable realistic programs to be written and to automatically check that they obey some confidentiality or integrity policy.

Web site: http://cristal.inria.fr/~simonet/soft/flowcaml/.

5.2.2. GCaml

Participants: Jun Furuse [U. Tokyo], Pierre Weis.

GCaml is an experimental extension of Objective Caml with extensional polymorphism, i.e., type-indexed functions that dispatch at run-time on the types of their arguments. Based on Jun Furuse's PhD thesis [42].

Web site: http://cristal.inria.fr/~furuse/generics/.

5.2.3. The Zen computational linguistics toolkit

Participant: Gérard Huet.

Zen is a toolkit for computational linguistics developed in Objective Caml by Gérard Huet. It powers Gérard Huet's Sanskrit Site, at http://cristal.inria.fr/~huet/SKT/.

Web site: http://cristal.inria.fr/~huet/ZEN/.

5.2.4. Htmlc

Participant: Pierre Weis.

Htmlc is an HTML template file expander that produces regular HTML pages from source files containing generated text fragments. These fragments can be the results of variable expansions, the output of an arbitrary Unix command (for instance, the last modification date of a page), or shared HTML files (such as a common page header or footer). Htmlc offers a server-independent way of defining templates that factor out the repetitive parts of HTML pages.

Web site: http://pauillac.inria.fr/htmlc/.

6. New Results

6.1. Type systems

6.1.1. Extending ML with (impredicative) second-order types

Participants: Didier Le Botlan, Didier Rémy, Boris Yakobowski.

The ML Language uses simple types (first-order types without quantifiers) enriched with type schemes (simple types with outer-most universal quantifiers). This allows for type inference based on first-order unification, relieving the user from the burden of writing type annotations. However, it only enables a limited form of parametric polymorphism. In contrast, System F uses second-order types (types with inner universal quantifiers at arbitrary depth) that are much more expressive. As a result, type inference is undecidable in System F. Therefore, the user must provide all type annotations.

Didier Le Botlan and Didier Rémy have proposed a new type system, called MLF, that still enables type synthesis as in ML while retaining the expressiveness of System F. MLF generalizes a previous proposal by Jacques Garrigue (Kyoto University) and Didier Rémy, called Poly-ML [2], which is now part of the OCaml type system. In Poly-ML and in OCaml, second-order types are embedded into first-order types by means of semi-explicit coercions. However, coercions draw a barrier between inferred types and explicit second-order types. This gap disappears in MLF because its type schemes subsume both ML type schemes with implicit instantiation, and System-F second-order types with inner polymorphism. The trick is to allow bound variable of type schemes to stand for another type scheme (providing for inner quantification) or to range over all instances of another type scheme (providing for ML-style type inference).

Remarkably, type inference in MLF reduces to a new form of unification that amounts to performing first-order unification in the presence of second-order types. All expressions of ML can be typed in MLF without any type annotations. All expresssions of System F can also be typed, but explicit type annotations are required for function parameters that are used polymorphically. The study of MLF was the topic of Didier Le Botlan's PhD dissertation [12], which he defended in May 2004. A small prototype implementation has also been written: http://cristal.inria.fr/~lebotlan/.

Boris Yakobowski, who started his PhD under Didier Rémy's supervision in October, is continuing the investigation of MLF. One goal is to build on the core MLF language and extends it toward a realistic programming language. In particular, the addition of primitives with side effects requires a restriction of the type system similar to the ML value restriction. The impact of this restriction on type inference remains to be investigated. The extension of MLF with recursive types, which would enable structural object types, also appears to be challenging. The simplification of the meta-theory of MLF is yet another direction for research. Finding a semantics for MLF types could simplify significantly the study of the properties of the language and makes the extensions above easier to formalize.

6.1.2. A framework for partial type inference in the predicative fragment of System F

Participants: François Pottier, Didier Rémy.

The predicative fragment of System F uses second-order types (that is, types with quantifiers at arbitrary depth) as in System F, but restricts the instantiation of type variables to first-order types. This system is somewhat closer to ML than to full System F, and hence makes partial first-order type inference easier. Indeed, by definition, type variables can only be instantiated by first-order types. François Pottier and Didier Rémy have proposed a core language where arguments of applications and let-bound expressions have explicit type annotations, so that type inference becomes a trivial ML-style type inference process based on first-order unification and let-polymorphism. In this system, monomorphic parts of types are inferred, while their polymorphic shapes are only checked and thus must be provided by the user. This clear separation can be exploited to propagate polymorphic shapes of user-provided annotations before type inference so as to reduce the amount of required type annotations. We use this approach to simplify bidirectional partial type inference for the predicative fragment of System F that has previously been proposed by Peyton-Jones and Shield. We decompose it into a simpler bidirectional elaboration process into the core language followed by ML-like type inference in the core language.

6.1.3. First-order canonical forms for second-order recursive types

Participants: Nadji Gauthier, François Pottier.

We studied a type system equipped with universal types and equirecursive types, which we refer to as F_μ. We showed that type equality may be decided in time $O(n \log n)$, an improvement over the previous known bound of $O(n^2)$. In fact, we showed that two more general problems, namely entailment of type equations and type unification, can be decided in time $O(n \log n)$, a new result. To achieve this bound, we associate with every F_μ type a *first-order canonical form* that can be computed in time $O(n \log n)$. By exploiting this notion, we reduced all three problems mentioned above to equality and unification of *first-order* recursive terms, for which efficient algorithms are known.

Preliminary results were presented at the APPSEM'04 workshop, and the paper appeared at ICFP'04 [22].

6.1.4. Extending ML with guarded algebraic data types

Participants: François Pottier, Yann Régis-Gianas, Vincent Simonet.

Guarded (or *generalized*) algebraic data types subsume the concepts known in the literature as *indexed types*, *guarded recursive datatype constructors* and *first-class phantom types*, and are closely related to *inductive types* as found in the Coq proof assistant. They have the distinguishing feature that, when typechecking a function defined by cases, every branch may be checked under different assumptions about the type variables in scope. This mechanism allows exploiting the presence of dynamic tests in the code to produce extra static type information.

Vincent Simonet and François Pottier have proposed an extension of the constraint-based type system HM(X) with deep pattern matching, guarded algebraic data types, and polymorphic recursion. They have proved that the type system is sound and that, provided recursive function definitions carry a type annotation, type inference can be reduced to constraint solving. This work has been submitted for publication [37].

In the specific case where only equality constraints are considered, this leads to a proposal for an extension of ML with guarded algebraic data types. During his master's internship, Yann Régis-Gianas has implemented a prototype type-checker for such a programming language.

6.1.5. Applications of guarded algebraic data types

Participants: François Pottier, Yann Régis-Gianas.

Imagining new applications of guarded algebraic types is currently a hot and open area of investigation. The goals are to explore the expressive power of this new feature, to discover applications that are interesting for their own sake, and to ensure that our proposal for extending ML with guarded algebraic data types is indeed able to express the required idioms.

The LR parser generators that are bundled with many functional programming language implementations produce code that is type-unsafe, needlessly inefficient, or both. François Pottier and Yann Régis-Gianas have shown that, using generalized algebraic data types, it is possible to produce parsers that are well typed (so they cannot unexpectedly crash or fail) and nevertheless efficient. This is a useful result as well as a nice exercise in programming with generalized algebraic data types.

This work has been submitted for publication [36]. Yann Régis-Gianas has implemented a prototype parser generator and verified that the parsers thus produced are accepted by his prototype typechecker.

6.1.6. Term enumeration

Participants: Boris Yakobowski, J. B. Wells [Heriot-Watt University].

Term enumeration is a type-directed program synthesis that outputs terms of a desired goal typing (i.e., environment of type assumptions and result type) using the values and functions available in the type environment. A typical use case is to generate glue code that adapts one module interface to another, overcoming simple interface differences.

Using the Curry-Howard correspondence (propositions-as-types, proofs-as-programs), the problem of term enumeration can be transformed into a *proof enumeration* problem for an intuitionistic logic calculus. Provided proof counting in this calculus is computable, it is possible to write efficient algorithms for term enumeration. A suitable calculus has been found in which the number of proofs of a sequent is easy to compute, and which produces few redundant programs. These results were published in [31].

6.2. XML transformation languages

6.2.1. Parametric polymorphism for XML languages

Participants: Alain Frisch, Giuseppe Castagna [ENS Paris], Haruo Hosoya [Tokyo University].

XDuce and related languages propose type systems to deal with XML documents within functional programming languages. They adopts a structural point of view: XML types (DTDs) denote sets of values. This approach yields a rich and natural notion of subtyping based on set inclusion, and the associated polymorphism based on subsumption. The CDuce language also offers another kind of polymorphism through overloaded functions.

This year, we have studied the addition of a third kind of polymorphism to XDuce and CDuce: parametric polymorphism in the style of the ML language. The challenge consists in adding type variables to the type algebra while sticking to the set-theoretic approach for subtyping and retaining practical typing algorithms. We introduced a technique of marked values to capture the intuition that type variables actually denote place-holders in values. We established a formal connection between the inference of types when calling a polymorphic function and the classical operation of pattern matching in XDuce. We obtained a language which is an extension of XDuce with parametrically-polymorphic functions and implicit instantiation. The corresponding type-cheking algorithms are not significantly more complex than the monomorphic ones.

These results will be presented at POPL 2005 [25].

6.2.2. Extending Caml with XML types

Participant: Alain Frisch.

The type system of Objective Caml extends the original Hindley-Milner type system in several directions, but keeps principal type inference based on first-order unification.

Various type systems for manipulation XML documents in functional languages have been proposed recently. They usually builds on an interpretation of types as sets of values, which induces a natural subtyping relation. Unlike Hindley-Milner type systems, they do not feature full type inference: function argument and return types have to be given by the programmer. Instead, they rely on tree automata algorithms to compute the subtyping relation and to propagate precise types through complex operations on XML documents such as regular expression pattern matching, deep iterations, path navigation.

In order to facilitate the manipulation of XML documents in Caml, we consider the integration of some of the features from XML transformation languages into Objective Caml. However, it seems quite challenging to merge in a single type systems ML-like type inference and XML types based on tree automata. A natural approach would be to turn to the HM(X) constraint-based formulation of the ML type system . Unfortunately, the constraint domain naturally arising from embedding XML types into HM(X) is undecidable because it can express the inclusion of context-free languages. Moreover, turning to a constraint-based approach might not be desirable from a practical point of view: issues arise concerning efficiency, readability of error messages and displayed types, and integration with existing implementations.

Our approach is to start from ideas proposed by Garrigue on structural polymorphism, that is, to attach local and fully-solved constraints to type variables (solved constraints cannot relate different type variables). Polymorphic variants in Objective Caml demonstrate that this kind of type system can be effectively implemented on top of existing ML implementation. Alain Frisch studied a generic way to extend ML with ground types taken from an external type algebra where types form a lattice (for an implicit subtyping relation). XML types are an instance of this theory.

6.3. Certified compilation

Formal methods are increasingly being applied to safety-critical software in order to increase their reliability and meet the requirements of the highest levels of software certification. However, formal methods are applied to the source code of the software, written in C or higher-level languages; but what actually runs in the safety-critical computer is the machine code generated from the sources by a compiler. Making sure that the compiler does not introduce bugs is a difficult process: extensive testing of the executable code can help, but the highest levels of certification in e.g. avionics actually require manual inspection of the assembly code produced by the compiler, which is painful and costly.

A better solution to this problem is to apply formal methods to the compiler itself. In the context of the coordinated research action (ARC) Concert (which involves projects Cristal, Lemme, Mimosa, Miró and Oasis, as well as CNAM), we are currently exploring the feasability of developing an *certified compiler*, that is, a realistic compiler that comes with a machine-checked proof that the generated assembly code has the same semantics as the source code.

6.3.1. The Concert experimental compiler and its certification

Participants: Xavier Leroy, Sandrine Blazy, Benjamin Grégoire [project Lemme], Laurence Rideau [project Lemme].

As part of the Concert explorations, Xavier Leroy developed a reference optimizing compiler from Cminor (a low-level imperative language in the style of C) to PowerPC assembly code. This compiler is written in the purely functional subset of OCaml and serves as a blueprint for the formalisation and proof effort, which uses the Coq proof assistant.

Sandrine Blazy, Benjamin Grégoire, Xavier Leroy and Laurence Rideau used the Coq proof assistant to specify operational semantics for the source, target and several intermediate languages (register transfer languages with control represented either as a flow graph or via explicit branches and labels). Benjamin Grégoire proved (in Coq) semantic preservation for two optimization passes based on dataflow analysis: constant propagation and common subexpression elimination. (See the activity report of project Lemme.)

Xavier Leroy proved semantic preservation for two translation phases. The first phase translates Cminor to a control-flow graph of register transfer instructions. It performs recognition of complex instructions, allocation of temporary pseudo-registers and construction of the flow graph. The second phase exploits the result of a register allocation pass. It maps temporary pseudo-registers to hardware registers and stack locations and enforces function calling conventions. The two developments represent about 5000 and 2500 lines of Coq, respectively, and required 4 man.months. The first phase was especially difficult to express and prove correct in Coq because it requires extensive error handling and generation of fresh temporaries and flow graph nodes. To express error handling and state management in terms of the pure, total functions offered by Coq, Xavier

Leroy used a monadic programming style inspired by Haskell, and developed ad-hoc proof tactics to reason about monadic computations in Coq.

6.3.2. Formal semantics of memory management

Participant: Sandrine Blazy.

Sandrine Blazy joined the Cristal project in september 2004 for a one-year sabbatical. She formalized and proved correct a memory model dedicated to a certified compiler. This model has been formalized at several levels of abstraction. At the more abstract level, the memory is defined as an abstract data type, *i.e.* by the operations (and their properties) that access and modify the memory. At the lowest level, an implementation of the memory has been defined for the Concert experimental certified compiler described above.

OCaml code has been extracted from this implementation in Coq. This work represents about 7000 lines of Coq and required 4 man-months. The results of this experiment show that the extraction of real-size OCaml code for Coq specifications is possible.

6.3.3. A semantic study of Static Single Assignment form

Participant: Boris Yakobowski.

During his DEA internship, Boris Yakobowski studied the use of the SSA (Single Static Assignment) form in the context of certified compilation. The SSA form is an intermediate compilation language in which every variable is assigned only once statically. It enables simpler and algorithmically more efficient compiler optimisations such as liveness analysis, dead-code elimination, common subexpressions elimination, ...

Boris Yakobowski obtained a new characterization of the shape of programs in SSA form. This characterization ensures that the usual properties of programs in SSA form found in the literature are verified. At the same time, it is not overly restrictive (in particular, most of the existing programs can be mechanically transformed into this form) and allows simple reasoning for proving program transformations.

Using these preliminary results, the proofs of correctness of all the optimizations considered in the Concert compiler have been done on paper. They appear to be simpler than the proofs already formalized in Coq without using SSA. The optimization algorithms themselves appear to be much more efficient and easier to write in Coq. It remains to be seen whether the proofs can be translated to Coq easily. A full report is available [38].

6.3.4. Development of a certifiable C compiler front-end

Participants: David Haguenauer, Xavier Leroy.

During his DEA (master's) internship, David Haguenauer designed and implemented in OCaml a front-end for the Concert reference compiler that translates a large subset of the C language into the Cminor input language of the Concert compiler. The medium-term goal is to obtain a certified compiler for a subset of C, usable to compile embedded software. David Haguenauer's experiment confirms that Cminor is broadly adequate as a first intermediate language for the compilation of C, but suggests several changes to Cminor that makes it better suited for this purpose.

6.4. Software-proof codesign

6.4.1. Focal

Participants: Damien Doligez, Pierre Weis, Richard Bonichon.

Focal — a joint effort with the LIP6 laboratory (U. Paris 6) and the CEDRIC laboratory (CNAM) — is a programming language and a set of tools for software-proof codesign. The most important feature of the language is an object-oriented module system that supports multiple inheritance, late binding, and parameterisation with respect to data and objects. Within each module, the programmer writes specifications, code, and proofs, which are all treated uniformly by the module system.

Focal supports the modular and incremental design of certified software libaries: starting from a set of pure-specification modules, the programmer uses inheritance to add code and proofs in a structured way. This

system promotes reusability by making it easy to use a set of abstract and semi-abstract modules to embody the general aspects of the problem at hand, before going into more specialised coding.

The Focal compiler takes focal source code, does type checking and dependency checking to ensure the consistency of inheritance and parameterisation, and produces two outputs: O'Caml code and an incomplete Coq proof script. The O'Caml code can be compiled and executed; it corresponds to the computational part of the Focal source. The Coq script must be completed before it can be passed to Coq for verification.

Focal proofs are done in a hierarchical language invented by Leslie Lamport in 1994. Each leaf of the proof tree is a lemma that must be proved before the proof is detailed enough for verification by Coq. The Focal compiler translates this proof tree into an incomplete proof script. This proof script is then completed by `zenon`, the automatic prover provided by Focal. `zenon` is a tableau-based prover for first-order logic with equality. It is developped by Damien Doligez with the help of David Delahaye (CNAM).

A beta version of Focal was released in October, comprising the `focc` compiler, the `zenon` prover, a library of computer algebra functions, and a tool for producing documentation in XML format.

6.5. The Objective Caml system, libraries and tools

6.5.1. The Objective Caml system

Participants: Xavier Leroy, Jacques Garrigue [Kyoto university], Damien Doligez, Maxence Guesdon, Luc Maranget [project Moscova], Michel Mauny, Pierre Weis.

This year, we released version 3.08 of the Objective Caml system, along with two updates 3.08.1 and 3.08.2. Damien Doligez acted as release manager for these three versions. The main novelties in these releases are:

- The introduction of immediate objects, i.e. objects defined by listing directly their methods and fields, without going through a class. This extension enables certain uses of objects to be expressed in a much more lightweight and concise manner.
- The internal representation of objects was simplified, enabling a more efficient compilation of class operations and the marshaling (serialization) of objects between identical programs.
- The lowest layer of the library for arbitrary-precision integer and rational arithmetic was rewritten from scratch to work around licensing problems.
- The runtime system was extensively cleaned up to prevent name collisions when linking with third-party C code.

In addition, a number of minor bugs and irregularities in the standard libraries were corrected.

6.5.2. Type-safety of unmarshaled OCaml values

Participants: Grégoire Henry, Michel Mauny.

Unmarshaling (reading OCaml values from disk files, for instance) is not a type-safe operation in the current OCaml implementation. Indeed, there is currently no way to provide an unmarshaling function with a polymorphic (parametric) type. Furthermore, marshaled values written to disk by OCaml do not carry type information. Generally, unmarshaling functions in staticallly typed programming languages have monomorphic types and return values containing explicit type information. Some form of dynamic type checking is then necessary (as a programming language construct) in order to recover a statically typed value from such an unmarshaled value.

Grégoire Henry and Michel Mauny studied the possibility of having unmarshaling functions that receive a representation of some type t as argument and return values of type t. This is obtained by extending the programming language with a predefined guarded data type: a parameterized abstract type α *tyrepr* whose values are representations of the current instance of the type parameter α. With such a mechanism, the representation of type t has type t *tyrepr*. Parameterized by a representation of type t, an unmarshaling function is then able to check that the value being unmarshaled indeed belongs to type t. When, as it is the case in OCaml, no type information is attached to marshaled values, the verification can be performed *via* a recursive

traversal of the value, with particular care brought to sharing and polymorphism, to mutable data structures, and to cycles.

Grégoire Henry and Michel Mauny extended this work to marshaled values containing closures (with code addresses): in this case, some cooperation is needed from the OCaml compiler.

This approach has been prototyped as an extension to OCaml (without support for closures) by Grégoire Henry.

6.5.3. OCamlJIT

Participant: Basile Starynkevitch.

OCamlJIT is an execution environment for OCaml bytecode that uses just-in-time compilation (JIT) instead of interpretation of the bytecode. That is, the bytecode is translated on the fly to unoptimized machine code, which is then executed without interpretation overhead. OCamlJIT builds on the GNU Lightning library for run-time machine code generation. OCamlJIT typically executes code twice as fast as the OCaml bytecode interpreter; actual speedups vary between 1 and 5. OCamlJIT provides the exact same services as the OCaml bytecode interpreter with the same interface, and can therefore execute directly all existing bytecoded Caml programs without recompilation. In particular, it supports JIT compilation of dynamically-loaded or dynamically-generated bytecode, as used for instance in the MetaOCaml extension of OCaml with staged computations. OCamlJIT was presented at the MetaOCaml workshop [30].

6.5.4. The Caml development environment

Participants: Maxence Guesdon, Pierre-Yves Strub [ingénieur expert MIRIAD].

The Cameleon integrated development environment for Objective Caml, described in section 5.1.5, was almost entirely ported to the LablGTK2 graphical user interface. The inteface builder was removed to use the Glade interface builder and the Lablgladecc code generator. The DBForge database interface was also improved to allow multicolumn indexes and foreign keys.

Maxence Guesdon developed Caml-get, a tool that facilitates the reuse of Caml source code that is not packaged as libraries. Caml-get automates the extraction of Caml definitions from reference source files and their insertion inside the sources of other projects, keeping track of later modifications on the reference source code. An integration of this tool in Cameleon as a plug-in is in progress.

6.5.5. XML-Light

Participants: Michel Mauny, Hanfei Wang.

XML-Light (http://tech.motion-twin.com/xmllight/) is a lightweight OCaml library that provides a minimal XML parser, accepting only a subset of XML. Hanfei Wang and Michel Mauny extended and completed this library: it now covers most of the XML specification, includes a SAX-like parser delivering streams of events, and provides two algorithms for validating XML documents against DTDs.

6.6. Coupling numerical codes using OCamlP3L

Participants: Pierre Weis, Roberto Di Cosmo [university Paris 7], François Clément [project Estime], Jérôme Jaffré [project Estime], Zheng Li [university Paris 7].

See the activity report of project Estime.

6.7. Computational linguistics

6.7.1. Computational linguistics

Participant: Gérard Huet.

Gérard Huet continued his work on developing a computational linguistics platform adapted to Sanskrit, based on applicative programming in Ocaml.

On the morphology front, he implemented the verbal conjugation paradigms, a rather complex task, since Sanskrit verbal forms admit 3 persons, 3 numbers (singular, dual and plural), 3 voices (active, middle and passive), 3 moods (indicative, imperative, optative) a present system with 10 families for tenses present and imperfect, a future system, a perfect system with a complex reduplicating scheme, an aorist system with 7 paradigm classes. Furthermore many roots admit derived stems for causative, intensive, and desiderative moods. This conjugation machine was instantiated on his digital lexicon, leading 75000 forms for the 515 roots. These forms can be compressed as a 54KB lexical dag, which yields in its turn a segmenting automaton of 120KB. This supplements the 144000 substantive declension forms (167KB dag, 921KB automaton). The segmenter/tagger he previously developed for substantive compounds is now able to segment small sentences, with a 3-pass method using 3 automata (substantives, roots, and preverbs).

The completion of this task induced a complete rehaul of the phonetics layer. In particular, two new phonemes were added, corresponding to two origins in Indo-European of Sanskrit phonemes 'h' and 'j'. With this finer phonemic distinctions, internal sandhi euphony rules could be written in a uniform and deterministic manner. These phonemes come in supplement to the two "phantom phonemes" which are necessary to account for associativity of external sandhi in the presence of the "ā" preverb [17].

This work basically concludes the morphological work. The Sanskrit processing site has been enriched with the 3-automaton segmenter/tagger, and a lexicon-directed lemmatizer. Full lists of inflected forms, in XML format, were released as free linguistic resources available for research purposes.

A structural classification of compounds was effected by machine, in order to complete the segmenting automaton [17] with compound forms which cannot be generated by standard euphony (external sandhi) from their components. This is a typical application of using programs to generate a systematic linguistic model from a corpus (here the lexicon). Such analysis was previously unavailable for the Sanskrit language, because of its complex morpho-phonetics structure.

A collaboration with Brendan Gillon, a linguist from Mc Gill University, was started on the syntax of Sanskrit. Brendan Gillon manually constructed over the years a corpus of Sanskrit sentences, issued from a classical treatise on Sanskrit syntax by Apte. From its exemple citations, Pr Gillon built an annotated phrase structure tree bank of over 500 sentences. G. Huet managed to parse the notation and reverse engineer this database into a fully structured tree bank. Work is under progress to adapt G. Huet segmenter so that each phrase structure item may be considered an enrichment of the consistent tagged representation. The next step will be to use this annotated corpus as training data for a parser, whose derivation trees will be consistent with the syntactic annotations.

Pr. Gillon visited our site of Rocquencourt for a week in June in order to jointly work out a more explicit syntactic notation, where long distance dependencies and anaphora antecedent links will be explicity represented in a notation for a forest of dislocated syntax trees. The various tree operations will correspond to elementary moves of a parser, which will enforce constraints issued from agreement and subcategorization conditions. This constraint machine is still in its design phase.

The design of the lexical database and its Web interface was presented in Geneva in August at the Workshop on Enhancing and Using Electronic Dictionaries, as part of the International Conference on Computational Linguistics (COLING 2004) [27].

6.7.2. A new applicative model for finite state machines

Participant: Gérard Huet.

This work builds on the Zen toolkit [43] for lexical processing designed by Gérard Huet, and distributed as a free software OCaml library. It investigates a notion of mixed automaton or *aum*, first presented in Huet's Automata Mista article [26]. This work is being pursued as a general model for the modular construction of finite state machines, possibly non-deterministic, and possibly transducing their input on an output tape, in a purely applicative inductive data type whose operations model constructions of regular relations.

7. Contracts and Grants with Industry

7.1. The Caml Consortium

The Caml Consortium is a formal structure where industrial and academic users of Caml can support the development of the language and associated tools, express their specific needs, and contribute to the long-term stability of Caml. Membership fees are used to fund specific developments targeted towards industrial users. Members of the Consortium automatically benefit from very liberal licensing conditions on the OCaml system, allowing for instance the OCaml compiler to be embedded within proprietary applications. Currently, four companies are members of the Caml Consortium: Artisan Components, Athys (a subsidiary of Dassault Systèmes), Dassault Aviation, and LexiFi. For a complete description of this structure, refer to http://caml.inria.fr/consortium/.

7.2. Licensing of the MMS software

MMS (Modular Module System) is a reusable implementation of a type-checker and elaborator for the OCaml module system. It enables to add an OCaml-like module system on top of an almost arbitrary base language. A license allowing commercial exploitation of this software was purchased by Athys, for use with a domain-specific language that is part of Dassault Systèmes's Catia environment for mechanical design.

8. Other Grants and Activities

8.1. National initiatives

The Cristal and Moscova projects participate in an *Action Concertée Incitative* GRID named "PL4-CARAML", also involving the PPS lab (University Paris 7), the LIFO (University of Orléans) and the LACL (University Paris 12). The theme of this action is the design of extensions to functional programming languages in order to program efficient parallel computations, based on the BSP parallel programming model. Our participation to this action focuses on the OCaml-related aspects: design of OCaml extensions and support for OCaml. This ACI is managed by the LIFO.

The Cristal, Logical, Miró and Protheo projects participate in an *Action Concertée Incitative "Sécurité et Informatique"* (ACI SI) named "Modulogic", coordinated by the LIP6 laboratory at university Paris 6 and also involving the Cedric laboratory at CNAM. The theme of this action is the design of a development environment for certified software, emphasizing modular development and proof. Our participation to Modulogic is centered around the Focal language (see section 6.4.1), with the long-term goals of incorporating rewriting into Focal and studying how to apply the Focal methodology to security-sensitive applications.

8.2. European initiatives

The Cristal project is involved in the European Esprit working group 26142 named "Applied Semantics II". The purpose of this working group, and its predecessor "Applied Semantics", is to foster communication between theoretical research in semantics and practical design and implementation of programming languages. This group is coordinated by the Ludwig-Maximilians University in Munich and comprises both European academic teams and industrial research labs. The yearly meetings of this working group allow us to keep good connections with the European semantics community.

The Cristal and Gemo projects participate in an European 6th Framework Programme STREP project named "Environment for the Development and Distribution of Open Source Software" (EDOS). The main objective of this project is to develop technology and tools to support and streamline the production, customization and updating of distributions of Open Source software. This STREP is coordinated by INRIA Futurs and involves both small and medium enterprises from the Open Source community (MandrakeSoft, SOT, ...) and academic research teams (U. Paris 7, U. Genève, Zurich U., Tel-Aviv U.).

9. Dissemination

9.1. Interaction with the scientific community

9.1.1. Learned societies

Gérard Huet is Member of the French Academy of Sciences.

Xavier Leroy and Didier Rémy are members of IFIP Working Group 2.8 (Functional Programming).

9.1.2. Collective responsibilities within INRIA

Xavier Leroy was chairman of the *Section locale d'Auditions* (local hiring committee) for the INRIA Rocquencourt CR2 hiring competition (64 candidates for 4 positions).

Michel Mauny is a member of the management team of the INRIA Rocquencourt research unit, delegate for scientific affairs. Until september 2004, he was *vice-président du Comité des Projets* (deputy chairman of the Projects Committee) and *membre suppléant nommé de la Commission d'Évaluation* (appointed deputy member of the Evaluation Committee). He was scientific organizer of the *rencontres INRIA-Industrie* 2004 (INRIA-Industry meeting), whose theme was Software Engineering. He participated in the hiring committee for *Techniciens de la Recherche* (administrative staff).

Pierre Weis is a member of the *comité d'UR de Rocquencourt*. He is *membre titulaire élu* (elected member) of the *Comité Technique Paritaire* and the *Comité de Concertation*, and *membre suppléant élu* (elected deputy member) of the *Commission d'Évaluation*.

9.1.3. Collective responsibilities outside INRIA

Sandrine Blazy is a member of the *Commissions de spécialistes* (hiring committees) of CNAM and University of Évry.

Xavier Leroy is a member of the steering committees of the ACM Symposium on Principles of Programming Languages (POPL) and of the Asian Association for Foundation of Software (AAFS). He is scientific advisor for the European IST Network DART (Dynamic Assembly, Reconfiguration and Type-checking).

Michel Mauny is a member of the executive office of the *Réseau National de Recherche en Télécommunications* (RNRT). He is a member of the scientific board of the *Groupement de recherche Algorithmique, Langage et Programmation* (GDR ALP). He chairs the scientific board of the French-Moroccan cooperation program *Réseaux STIC*.

Pierre Weis is on the steering comittee of the JFLA conference.

9.1.4. Editorial boards and program committees

Xavier Leroy is associate editor of the Journal of Functional Programming. Didier Rémy is member of the editorial board of the Journal of Functional Programming.

Gérard Huet was member of the program committee of the 11th Workshop on Logic, Language, Information and Computation (WoLLIC'2004).

Xavier Leroy participated in the program committees of the Smart Card Research and Advanced Application IFIP Conference (CARDIS 2004) and of the European Symposium on Programming (ESOP 2005).

François Pottier was a member of the program committees for the 32nd ACM Symposium on Principles of Programming Languages (POPL 2005) and the 2005 conference on Types in Language Design and Implementation (TLDI 2005).

Didier Rémy participated in the program committee of the Journées Francophones des Langages Applicatifs (JFLA 2005).

9.1.5. PhD and habilitation juries

Gérard Huet was reviewer (*rapporteur*) of the habilitation of Solange Coupet (U. de Provence, september 2004) and of the habilitation of Claire Gardent (U. Nancy 1, september 2004).

Xavier Leroy was external examiner for Robert Ennals's PhD thesis (Cambridge University, april 2004). He chaired the PhD jurys of Francesco Logozzo (École Polytechnique, 15 june 2004) and Pierre Letouzey

(U. Paris Sud, 9 july 2004). He participated in the habilitation jury of David Naccache (U. Paris 7, december 2004) and in the PhD jury of Alain Frisch (U. Paris 7, december 2004).

François Pottier was reviewer (*rapporteur*) for Emmanuel Coquery's Ph.D. thesis (U. Paris 6, december 2004). He participated in the PhD jury of Vincent Simonet (U. Paris 7, march 2004).

Didier Rémy was a member of the PhD jury of his students Alexandre Frey (École des Mines, june 2004) and Didier Le Botlan (École Polytechnique, may 2004).

9.1.6. The Caml user community

Maxence Guesdon maintains the Caml Humps (http://caml.inria.fr/humps/), a comprehensive Web index of about 450 Caml libraries, tools and tutorials contributed by Caml users. This Web site contributes significantly to the visibility of the Caml language.

9.2. Teaching

9.2.1. Supervision of PhDs and internships

Alain Frisch co-supervised the master's internship of Kim Nguyen (5 months) together with Giuseppe Castagna (ENS).

Xavier Leroy supervised the master's internships of David Haguenauer and Boris Yakobowski.

François Pottier supervises Nadji Gauthier's Ph.D. studies. François Pottier supervised the master's internship of Yann Régis-Gianas over a period of six months. Still under François' supervision, Yann is now pursuing a Ph.D.

Michel Mauny supervised Grégoire Henry's summer internship (first-year master's student). He supervised Hanfei Wang (associate professor, Wuhan University, visiting scholar sponsored by the Chinese Government) from november 2003 to november 2004.

Didier Rémy supervised the PhD work of Alexandre Frey (defended in june 2004) and Didier Le Botlan (defended in may 2004). He supervises the PhD work of Daniel Bonniot and, since october 2004, Boris Yakobowski.

Pierre Weis supervised the DESS internship of Burhan Ulhaq and the engineering internship of Gaëlle Le Page.

9.2.2. Graduate courses

The Cristal project-team is strongly involved in the *Master Parisien de Recherche en Informatique* (MPRI), a graduate curriculum co-organized by University Paris 7, École Normale Supérieure Paris, École Normale Supérieure de Cachan and École Polytechnique. This master's program succeeds the DEA *Programmation* and the DEA *Algorithmique* previously organized by these institutions. In the context of the MPRI, Gérard Huet taught a 15-hour course entitled *Calculs Algébriques et Fonctionnels* (fundamental theorems of λ-calculus). François Pottier taught a 18-hour course entitled *Langages de programmation: sémantique et typage* (type systems and operational semantics). Xavier Leroy participated in the setup of the MPRI and represents INRIA on its board of directors.

Xavier Leroy gave a 6-hour lecture on abstract machines and compilation of functional languages at the *École Jeune Chercheurs en Programmation* (Nantes, may 2004), a summer school attended by about 30 first-year PhD students.

François Pottier taught a 5 hour course entitled *Type-based information flow analyses* at the Summer School on Software Security, Eugene, Oregon.

François Pottier and Didier Rémy have contributed a chapter on the essence of ML type inference to a graduate textbook edited by Benjamin Pierce [18].

9.2.3. Undergraduate courses

Michel Mauny gave two courses to engineering students, one on functional programming at ISIA (20 hours), the other on the Unix system at ETGL (12 days).

Didier Rémy is a part-time professor at École Polytechnique. This year, he taught a course on "Operating Systems: principles and programming" to third year students. Maxence Guesdon was teaching assistant for this course.

9.3. Participation in conferences and seminars

9.3.1. Participation in conferences

Nadji Gauthier, Xavier Leroy, Michel Mauny, François Pottier and Vincent Simonet attended the symposium Principles of Programming Languages and the satellite workshop Foundations of Object-Oriented Languages (Venice, Italy, january 2003). François Pottier presented a paper on polymorphic typed defunctionalization co-authored with Nadji Gauthier [29].

Xavier Leroy, Michel Mauny and Pierre Weis attended the Journées Francophones des Langages Applicatifs (Ile de Ré, France, january 2004). Xavier Leroy gave an invited lecture on formal certification of optimizing compilers.

Xavier Leroy was keynote speaker at the workshop on Construction and Analysis of Safe, Secure and Interoperable Smart Devices (Marseilles, France, march 2004). He lectured on exploiting type systems and static analyses for smart card security.

Michel Mauny attended the yearly colloquium of the RNRT network (Metz, France, march 2004).

Nadji Gauthier, Xavier Leroy and Didier Rémy attended the workshop on Applied Semantics (Tallinn, Estonia, april 2004). Nadji Gauthier presented preliminary results about second-order recursive types, later published as [22]. Xavier Leroy gave a short communication on the compilation of generalized recursive definitions [24].

Vincent Simonet was invited to participate in the International Workshop on Formal Methods and Security in Nanjing (China) in May 2004. He talked about the Flow Caml system.

Damien Doligez attended the International Joint Conference on Automated Reasoning (Cork, Ireland, july 2004).

Alain Frisch attended the Colloquium on Automata, Languages and Programming (ICALP) (Turku, Finland, july 2004) where he presented a paper on the greedy disambiguation policy for regular expression pattern matching [20].

Boris Yakobowski participated in the International Symposium on Logic-based Program Synthesis and Transformation (Verona, Italy, august 2004), where he presented his work on term enumeration [31].

Alain Frisch attended the International Conference on Theoretical Computer Science (TCS) (Toulouse, France, august 2004) where he presented a paper on compilation techniques for efficient pattern matching in CDuce [21].

In august 2004, Gérard Huet delivered one of the evening lectures at the ESSLLI'2004 summer school in Nancy, on the topic "Splitting the Entropy Gordian Knot". He also attended the ACL'2004 conference in Barcelona. At the EuroScience Forum 2004 in Stockholm, he gave a lecture on "Functionality theory applied to Informatics, Logic and Linguistics". Finally, he talked on "Design of a Lexical Database for Sanskrit" at the workshop on Enhancing and Using Electronic Dictionaries, part of the Coling 2004 conference in Geneva.

Nadji Gauthier and Yann Régis-Gianas attended the International Conference on Functional Programming (Snowbird, Utah, september 2004). Nadji Gauthier presented his paper on second-order recursive types co-authored with François Pottier [22].

Xavier Leroy attended the workshop on Unconventional Programming Paradigms (Mont-Saint-Michel, France, september 2004).

Sandrine Blazy, Damien Doligez and Boris Yakobowski attended the 2nd workshop on Coq and Rewriting (Palaiseau, France, september 2004). Damien Doligez gave an invited presentation of the `zenon` prover.

Xavier Leroy and Michel Mauny participated in the first workshop of the Hyper-Learning GDRE (Cortona, Italy, october 2004). Xavier Leroy gave a tutorial lecture on functional programming for Web applications. Michel Mauny gave a tutorial lecture on typed functional languages for XML processing.

Basile Starynkevitch attended the GPCE'04 conference and the satellite workshop on MetaOCaml (Vancouver, Canada, october 2004). At the workshop, he presented his paper on OCamlJIT [30].

Didier Rémy participated in the meeting of IFIP working group 2.8 "Functional Programming" (West Point, New York, november 2004). He presented his work with François Pottier on partial type inference in the predicative fragment of System F.

9.3.2. Invitations and participation in seminars

Alain Frisch was invited to present the CDuce language at the Meeting on Types organized by the REWERSE Network of Excellence (Paris, France, october 2004), and also at the IRIT Seminar (Toulouse, France, december 2004).

Gérard Huet was invited to speak at the Colloquium of INRIA Sophia-Antipolis in february 2004. He gave a talk on "La théorie de la fonctionnalité à la croisée des chemins entre Informatique, Logique et Linguistique". In March, he talked on "De Zen à Aum" at the seminar of the LIMSI CNRS lab in Orsay. In May, he gave a lecture at Chalmers University in Göteborg at the occasion of his reception of a Honoris Causa Doctorate of Technology. He spoke on "Computational Linguistics from Zen to AuM". In October he talked on "Transducteurs d'état fini applicatifs et traitement des langues naturelles" at the Colloquium CMAT-CMAP-STIX, École Polytechnique, Palaiseau. In November he gave a talk "Des Lambdas et des Aums" at the Conférence en l'honneur du Pr Martin-Löf, in Marseilles Luminy.

Michel Mauny went to Rabat (Morocco) in december 2004 to attend the annual meeting of the French-Moroccan program for scientific cooperation *Réseaux STIC*.

In november 2004, François Pottier and Yann Régis-Gianas paid a one week visit to Simon Peyton Jones at Microsoft Research in Cambridge, Great Britain. Both of them gave a talk.

Vincent Simonet presented his work about information flow analysis at the seminar of the LSV laboratory (ENS Cachan) in march 2004.

9.4. Industrial relations

Gérard Huet ported his Sanskrit processing workbench as an application for the Simputer, a hand-held computing device running Linux developed in India. In December he visited the PicoPeta corporation in Bangalore, one of the manufacturers of the Simputer, in order to initiate a possible technology transfer towards a pocket Sanskrit machine.

Xavier Leroy is consultant for the Trusted Logic start-up company, one day per week, in the context of a *convention de concours scientifique*.

Pierre Weis is scientific collaborator at LexiFi, a start-up company that use Caml to design and implement a domain-specific language devoted to formal description and pricing of financial contracts.

9.5. Other dissemination activities

A team including Damien Doligez, David Haguenauer, Grégoire Henry, Xavier Leroy, Yann Régis-Gianas and Boris Yakobowski won the Judge's Prize at the 7th ICFP Programming Contest.

Maxence Guesdon participated in the Solutions Linux exhibition (Paris, feb 4–6, 2004). At the INRIA booth, he presented Objective Caml to potential users and generally promoted the language.

With help from Guillaume Rousse (project Atoll) and François Clément (project Estime), Pierre Weis prepared the 2004 release of the CD-ROM "Free Software available at INRIA".

Maxence Guesdon took part in the design and development of the Web site of the *Association Libre Cours*, http://www.librecours.org/. This non-profit association collects freely available courses and lecture notes in French and ensures that they are widely available to students and teachers.

10. Bibliography

Major publications by the team in recent years

[1] G. COUSINEAU, M. MAUNY. *The Functional Approach to Programming*, Cambridge University Press, 1998.

[2] J. GARRIGUE, D. RÉMY. *Extending ML with semi-explicit higher-order polymorphism*, in "Information & Computation", vol. 155, n° 1/2, 1999, p. 134–169.

[3] T. HIRSCHOWITZ, X. LEROY. *Mixin modules in a call-by-value setting*, in "Programming Languages and Systems – ESOP'2002", D. LE MÉTAYER (editor)., Lecture Notes in Computer Science, vol. 2305, Springer-Verlag, 2002, p. 6–20, http://cristal.inria.fr/~xleroy/publi/mixins-cbv-esop2002.pdf.

[4] D. LE BOTLAN, D. RÉMY. *MLF: Raising ML to the power of System-F*, in "Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming", August 2003, p. 27–38, http://cristal.inria.fr/~remy/work/mlf/icfp.pdf.

[5] X. LEROY. *A modular module system*, in "Journal of Functional Programming", vol. 10, n° 3, 2000, p. 269–303, http://cristal.inria.fr/~xleroy/publi/modular-modules-jfp.ps.gz.

[6] F. POTTIER. *A Versatile Constraint-Based Type Inference System*, in "Nordic Journal of Computing", vol. 7, n° 4, 2000, p. 312–347.

[7] F. POTTIER. *Simplifying subtyping constraints: a theory*, in "Information & Computation", vol. 170, n° 2, 2001, p. 153–183.

[8] F. POTTIER, V. SIMONET. *Information Flow Inference for ML*, in "ACM Transactions on Programming Languages and Systems", vol. 25, n° 1, January 2003, p. 117–158, http://cristal.inria.fr/~fpottier/publis/fpottier-simonet-toplas.ps.gz.

[9] D. RÉMY. *Using, Understanding, and Unraveling the OCaml Language*, in "Applied Semantics. Advanced Lectures", G. BARTHE (editor)., Lecture Notes in Computer Science, vol. 2395, Springer-Verlag, 2002, p. 413–537.

[10] P. WEIS, X. LEROY. *Le langage Caml*, second, Dunod, July 1999.

Doctoral dissertations and Habilitation theses

[11] A. FREY. *Approche algébrique du typage d'un langage à la ML avec objets, sous-typage et multi-méthodes*, Ph. D. Thesis, École des Mines de Paris, June 2004.

[12] D. LE BOTLAN. *MLF: Une extension de ML avec polymorphisme de second ordre et instanciation implicite.*, Ph. D. Thesis, École Polytechnique, May 2004, http://www.inria.fr/rrrt/tu-1071.html.

[13] F. POTTIER. *Types et contraintes*, Mémoire d'habilitation à diriger des recherches, Université Paris 7, December 2004, http://cristal.inria.fr/~fpottier/publis/fpottier-hdr.pdf.

[14] V. SIMONET. *Inférence de flots d'information pour ML: formalisation et implantation*, Ph. D. Thesis, Université Paris 7, March 2004, http://cristal.inria.fr/~simonet/publis/simonet-these.pdf.

Articles in referred journals and book chapters

[15] P. FLAJOLET, G. HUET. *Mathématiques et informatique*, in "Les mathématiques dans le monde contemporain", J.-C. YOCCOZ (editor)., Rapport sur la science et la technologie n°20, Académie des sciences, 2004.

[16] T. HIRSCHOWITZ, X. LEROY. *Mixin modules in a call-by-value setting*, in "ACM Transactions on Programming Languages and Systems", Accepted for publication, to appear, 2004, http://cristal.inria.fr/~xleroy/publi/mixins-cbv-toplas.pdf.

[17] G. HUET. *A Functional Toolkit for Morphological and Phonological Processing, Application to a Sanskrit Tagger*, in "Journal of Functional Programming", Accepted for publication, to appear, 2004, http://cristal.inria.fr/~huet/PUBLIC/tagger.pdf.

[18] F. POTTIER, D. RÉMY. *The Essence of ML Type Inference*, in "Advanced Topics in Types and Programming Languages", B. C. PIERCE (editor)., chap. 10, MIT Press, 2005, p. 389–489.

Publications in Conferences and Workshops

[19] V. BALAT, R. DI COSMO, M. FIORE. *Extensional Normalisation and Type-Directed Partial Evaluation for Typed Lambda Calculus with Sums*, in "31st ACM symposium on Principles of Programming Languages", ACM Press, January 2004, p. 64–76.

[20] A. FRISCH, L. CARDELLI. *Greedy regular expression matching*, in "31st International Colloquium on Automata, Languages and Programming", Springer-Verlag, July 2004, p. 618-629.

[21] A. FRISCH. *Regular tree language recognition with static information*, in "3rd IFIP International Conference on Theoretical Computer Science", Kluwer Academic Publishers, August 2004.

[22] N. GAUTHIER, F. POTTIER. *Numbering Matters: First-Order Canonical Forms for Second-Order Recursive Types*, in "Proceedings of the 2004 ACM SIGPLAN International Conference on Functional Programming (ICFP'04)", September 2004, p. 150–161, http://cristal.inria.fr/~fpottier/publis/gauthier-fpottier-icfp04.pdf.

[23] T. HIRSCHOWITZ, X. LEROY, J. B. WELLS. *Call-by-Value Mixin Modules: Reduction Semantics, Side Effects, Types*, in "European Symposium on Programming", Lecture Notes in Computer Science, vol. 2986, Springer-Verlag, 2004, p. 64–78, http://cristal.inria.fr/~xleroy/publi/mixins-mm-esop2004.ps.gz.

[24] T. HIRSCHOWITZ, X. LEROY, J. B. WELLS. *Extended recursive definitions in call-by-value languages, with applications to mixin modules and recursive modules*, in "Electronic proceedings of the second APPSEM II Workshop", April 2004.

[25] H. HOSOYA, A. FRISCH, G. CASTAGNA. *Parametric Polymorphism for XML*, in "32nd ACM symposium on Principles of Programming Languages", To appear, ACM Press, January 2005.

[26] G. HUET. *Automata Mista*, in "Verification: Theory and Practice: Essays Dedicated to Zohar Manna on the

Occasion of His 64th Birthday", N. DERSHOWITZ (editor)., Lecture Notes in Computer Science, vol. 2772, Springer-Verlag, 2004, p. 359–372, http://cristal.inria.fr/~huet/PUBLIC/zohar.pdf.

[27] G. HUET. *Design of a Lexical Database for Sanskrit*, in "Workshop on Enhancing and Using Electronic Dictionaries, COLING 2004", International Conference on Computational Linguistics, 2004, http://cristal.inria.fr/~huet/PUBLIC/coling.pdf.

[28] G. HUET. *Lexicon-directed Segmentation and Tagging of Sanskrit*, in "XIIth World Sanskrit Conference, Linguistics volume", to appear, Motilal Banarsidass, Delhi, 2004.

[29] F. POTTIER, N. GAUTHIER. *Polymorphic Typed Defunctionalization*, in "31st ACM symposium on Principles of Programming Languages", ACM Press, January 2004, p. 89–98, http://cristal.inria.fr/~fpottier/publis/fpottier-gauthier-popl04.pdf.

[30] B. STARYNKEVITCH. *OCamlJIT: a faster Just-In-Time OCaml Implementation*, in "First MetaOCaml Workshop", ACM Press, 2004, http://cristal.inria.fr/~starynke/starynke_ocamljit_july2004.ps.

[31] J. B. WELLS, B. YAKOBOWSKI. *Graph-Based Proof Counting and Enumeration with Applications for Program Fragment Synthesis*, in "International Symposium on Logic-based Program Synthesis and Transformation 2004 (LOPSTR 2004)", S. ETALLE (editor)., Lecture Notes in Computer Science, vol. 3049, Springer-Verlag, 2004.

Internal Reports

[32] F. CLÉMENT, R. DI COSMO, Z. LI, V. MARTIN, A. VODICKA, P. WEIS. *Parallel Programming with the System Applications to Numerical Code Coupling*, Research report, n° 5131, INRIA, 2004, http://www.inria.fr/rrrt/rr-5131.html.

[33] X. LEROY, D. DOLIGEZ, J. GARRIGUE, D. RÉMY, J. VOUILLON. *The Objective Caml system, documentation and user's manual – release 3.08*, INRIA, July 2004, http://caml.inria.fr/ocaml/htmlman/.

Miscellaneous

[34] D. HAGUENAUER. *Pré-compilation d'un sous-ensemble du langage C*, Master's dissertation (mémoire de stage de DEA), Université Paris 6, September 2004.

[35] T. HIRSCHOWITZ, X. LEROY, J. B. WELLS. *Compilation of extended recursion in call-by-value functional languages*, submitted and resubmitted after revision to Higher-Order and Symbolic Computation, November 2004.

[36] F. POTTIER, Y. RÉGIS-GIANAS. *Towards efficient, typed LR parsers*, Submitted to the Journal of Functional Programming, September 2004, http://cristal.inria.fr/~fpottier/publis/fpottier-regis-gianas-04.pdf.

[37] V. SIMONET, F. POTTIER. *Constraint-Based Type Inference with Guarded Algebraic Data Types*, Submitted to ACM Transactions on Programming Languages and Systems, June 2004, http://cristal.inria.fr/~fpottier/publis/simonet-pottier-hmg.pdf.

[38] B. YAKOBOWSKI. *Étude sémantique d'un langage intermédiaire de type Static Single Assignment*, Master's dissertation (mémoire de stage de DEA), ENS Cachan, September 2004.

Bibliography in notes

[39] G. BRACHA. *The programming language Jigsaw: mixins, modularity and multiple inheritance*, Ph. D. Thesis, University of Utah, 1992.

[40] D. DOLIGEZ, X. LEROY. *A concurrent, generational garbage collector for a multithreaded implementation of ML*, in "Proc. 20th symp. Principles of Programming Languages", ACM press, 1993, p. 113–123, http://cristal.inria.fr/~xleroy/publi/concurrent-gc.ps.gz.

[41] C. FOURNET, L. MARANGET, C. LANEVE, D. RÉMY. *Inheritance in the join calculus*, in "Foundations of Software Technology and Theoretical Computer Science – FSTTCS 2000", Lecture Notes in Computer Science, vol. 1974, Springer-Verlag, 2000.

[42] J. FURUSE. *Extensional polymorphism: theory and applications*, Ph. D. Thesis, University Paris 7, December 2002.

[43] G. HUET. *Zen and the Art of Symbolic Computing: Light and Fast Applicative Algorithms for Computational Linguistics*, in "Practical Aspects of Declarative Languages (PADL) symposium", 2003, http://cristal.inria.fr/~huet/PUBLIC/padl.pdf.

[44] X. LEROY. *A syntactic theory of type generativity and sharing*, in "Journal of Functional Programming", vol. 6, n° 5, 1996, p. 667–698, http://cristal.inria.fr/~xleroy/publi/syntactic-generativity.ps.gz.

[45] F. ROUAIX. *A Web navigator with applets in Caml*, in "Proceedings of the 5th International World Wide Web Conference, in Computer Networks and Telecommunications Networking", vol. 28, Elsevier, May 1996, p. 1365–1371.

[46] D. RÉMY. *Type Inference for Records in a Natural Extension of ML*, in "Theoretical Aspects Of Object-Oriented Programming. Types, Semantics and Language Design", C. A. GUNTER, J. C. MITCHELL (editors)., MIT Press, 1993.

[47] D. RÉMY. *Programming Objects with ML-ART: An extension to ML with Abstract and Record Types*, in "Theoretical Aspects of Computer Software", M. HAGIYA, J. C. MITCHELL (editors)., Lecture Notes in Computer Science, vol. 789, Springer-Verlag, April 1994, p. 321–346.

[48] D. RÉMY, J. VOUILLON. *Objective ML: A simple object-oriented extension to ML*, in "24th ACM Conference on Principles of Programming Languages", ACM Press, 1997, p. 40–53.