INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet CRISTAL

Programmation typée, modularité et compilation

Rocquencourt

THÈME 2A

Table des matières

1 Composition de l'équipe

Responsable scientifique
Michel Mauny [DR, INRIA]
Responsable permanent
Pierre Weis [DR, INRIA]
Assistante de projet (en commun avec Logical)
Nelly Maloisel [TR]
Personnel INRIA
Roberto Di Cosmo [Prof. Paris 7 en détachement depuis 09/2000]
Gérard Huet [DR]
Xavier Leroy [DR]
François Pottier [CR]
Daniel de Rauglaudre [IR]
Didier Rémy [DR]
Bruno Verlyck [IR, temps partiel]
Ingénieur associé
Maxence Guesdon
Collaborateur extérieur
Robert Harley [Inscrit en doctorat à l'Université Paris 7]
Doctorants
Daniel Bonniot [AMN, université Paris 7]
Pascal Cuoq [Boursier INRIA, université Paris 6]
Jun Furuse [Boursier du gouvernement japonais, université Paris 7]
Benjamin Grégoire [Allocataire MENRT, université Paris 7, commun avec Logical]
Tom Hirschowitz [Allocataire MENRT, université Paris 7]
Didier Le Botlan [AMX, université Paris 7]
Vincent Simonet [Normalien, université Paris 7]
Stagiaire
Farhad Mehta [IIT Dehli]

2 Présentation et objectifs généraux

Le projet Cristal s'intéresse aux formalismes de typage des langages de programmation et étudie les méthodes qui sous-tendent leur conception et l'établissement de leurs propriétés. Nos travaux concernent aussi les modèles d'exécution des programmes et débouchent sur la conception et la mise en œuvre d'outils de programmation typée robustes et efficaces.

Le typage statique accroît la sécurité de la programmation, la rapidité du développement d'applications et facilite leur maintenance. Les systèmes de types figurent aussi parmi les formalismes principaux de recherche de preuves de programmes où les types sont vus comme des spécifications. Il s'agit là d'autant d'arguments qui montrent la pertinence et l'utilité d'environnements de programmation typée qui procurent fiabilité, sécurité et efficacité dans le développement.

Nos travaux se situent donc au carrefour de la théorie des types, de la conception et la mise en œuvre des langages de programmation et de la programmation proprement dite.

3 Fondements scientifiques

3.1 Systèmes de types

Glossaire :

Typage Méthodologie de programmation qui permet de vérifier et de garantir l'utilisation cohérente des données manipulées par les programmes.

Typage statique ou dynamique Lorsque la vérification de types est effectuée *avant* l'exécution du programme (généralement par le compilateur), on dit que le typage est *statique*. Lorsqu'elle a lieu pendant l'exécution, on dit que le typage est *dynamique*.

Synthèse de types On dit que les types des programmes sont *synthétisés* (ou encore *inférés*) lorsque le compilateur déduit de lui-même tout ou partie des types qui interviennent dans le programme, ce qui libère le programmeur de l'obligation de spécifier les types que ses programmes manipulent.

Polymorphisme Un type est dit *polymorphe* lorsque certaines de ses composantes sont des variables qui représentent toute une famille de types (obtenus par *instanciation* de ces variables, c'est-à-dire remplacement de ces variables par d'autres types).

Résumé :

Le typage est un outil fondamental de la programmation : il permet de spécifier et vérifier la cohérence de la manipulation des données par les programmes, et de définir les mécanismes de structuration du code qu'on utilise pour la construction d'applications complexes (fonctions ou procédures, objets, modules, surcharge).

L'équipe Cristal étudie les systèmes de types et les méthodes formelles correspondantes, avec pour objectif de permettre la conception de langages et d'outils de programmation, sûrs et aux propriétés formellement établies.

Les langages de programmation de haut niveau aident à structurer le code par des constructions (fonctions/procédures, objets, surcharge, modules) dont la sémantique est intimement liée aux systèmes de types. En particulier, les systèmes de types servent de support à la détection (statique ou dynamique) d'erreurs de programmation.

Le projet Cristal étudie, sous l'angle des systèmes de types, les fondements de la programmation fonctionnelle, impérative, modulaire et par objets. Nous nous intéressons aussi bien aux aspects statiques que dynamiques des langages typés.

3.1.1 Synthèse de types de ML

Le langage ML dispose, depuis sa conception en 1978, d'un système de types qui permet la synthèse automatique des types à la compilation, et qui dispose aussi d'une forme de polymorphisme qui autorise la programmation d'algorithmes génériques (c'est-à-dire d'algorithmes qui acceptent de traiter toute une variété de données de types différents). Le système de types

de ML est spécifié formellement, et un théorème énonce sa propriété principale : aucune erreur de type ne peut se produire durant l'exécution d'un programme bien typé.

De par sa conception et sa généralité, le système de types de ML a servi de cadre d'étude à de nombreux travaux sur le typage :

- de nombreuses extensions lui ont été apportées, afin d'accroître l'expressivité du langage, tout en préservant les propriétés théoriques essentielles,
- les méthodes développées à cette occasion ont souvent été réutilisées et enrichies pour s'appliquer à d'autres études éventuellement bien au delà du strict cadre du langage ML.

Nos recherches dans ce domaine, même si elles ont une vocation plus générale, trouvent souvent des applications dans le langage Caml, la version de ML conçue et développée dans notre équipe.

Dans le domaine des systèmes de types, les sujets que nous étudions actuellement vont des fondements de la programmation par objets à la sécurité des programmes, en passant par différentes formes de polymorphisme, et se prolongent dans l'utilisation des formalismes de typage pour des analyses statiques qui dépassent largement le cadre traditionnel des systèmes de types. Ces différents sujets sont détaillés dans les sections suivantes.

3.1.2 Types d'ordre supérieur

Le langage ML manipule des types de premier ordre (sans quantificateurs), enrichis par des schémas de types qui autorisent la quantification universelle, et fournissent par là le polymorphisme, mais seulement de façon externe. Cette quantification externe permet de synthétiser les types de façon simple, par un calcul d'unification de premier ordre, mais limite l'expressivité. À l'opposé, le système F (lambda-calcul typé) autorise des types de second ordre où les quantificateurs peuvent être imbriqués à une profondeur arbitraire, ce qui lui confère une plus grande expressivité, mais rend la synthèse de types indécidable. La recherche de formalismes de typage combinant les avantages des deux approches est un axe important dans lequel des résultats permettraient d'augmenter de façon notable l'expressivité des langages à la ML en renforçant leur capacité d'abstraction, favorisant la réutilisation de code.

3.1.3 Typage des objets

La programmation par objets connaît un succès important depuis plusieurs années, mais peu de langages disposent d'une sémantique clairement et formellement établie. Diverses approches ont été proposées avec pour but d'établir les fondements de la programmation par objets en termes de théorie des types.

L'une des approches que suit le projet Cristal repose sur les travaux de Didier Rémy, qui a proposé dans sa thèse un système d'enregistrements extensibles qui se prête bien à la synthèse de types. Après la conception et la réalisation d'une première maquette d'un système à objets pour ML appelé ML-ART[Rém94], Didier Rémy s'est attaqué à l'ajout d'objets dans le langage Caml, devenu ainsi Objective Caml (OCaml). Développée en collaboration avec Jérôme

[Rém94] 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 (éditeurs), *Lecture Notes in Computer Science, 789*, Springer-Verlag, p. 321–346, avril 1994.

Vouillon, cette couche à objets est compatible avec la synthèse de types polymorphes et s'appuie sur un système de classes et l'héritage multiple ; c'est l'une des approches les plus attrayantes qu'on ait proposées pour les langages fortement typés, en dépit de la contrainte supplémentaire que représente la synthèse des types. La puissance de ce système de types permet ainsi de typer des programmes qui sont réputés difficiles à typer, même dans les langages où l'information de types est explicite.

Actuellement, nos efforts se portent sur le typage des multi-méthodes (qui prennent en compte plusieurs de leurs arguments pour déterminer le code qui doit être exécuté), ainsi que sur l'adjonction d'objets dans les langages concurrents, fondés par exemple sur le calcul Join.

Le sous-typage implicite, qui a été étudié par François Pottier dans sa thèse, permettrait de rendre OCaml encore plus performant et convivial. Il fournit en outre des résultats suffisamment généraux pour aborder des problèmes autres que celui du typage (par exemple, l'optimisation ou les problèmes de sécurité).

Les systèmes de modules ont un certain nombre de points communs avec les systèmes à objets. Ils fournissent eux aussi un moyen efficace de réutilisation de code, mais plutôt sous la forme de bibliothèques dotées d'interfaces, et doivent pouvoir être compilés séparément, afin de permettre aux programmeurs de concevoir des applications dont les composants peuvent être développés indépendamment les uns des autres. Malgré ces différences essentielles, il serait souhaitable de pouvoir bénéficier au niveau des modules de possibilités fournies par les objets : héritage, redéfinition, récursion mutuelle, par exemple. Une voie actuellement étudiée dans l'équipe Cristal est celle des calculs de *mixins*, que l'on peut voir comme des calculs de base de systèmes de modules.

Enfin la dernière approche que nous explorons repose sur la surcharge d'identificateurs et les fonctions génériques, ces fonctions dont le comportement est guidé par le type avec lequel elles sont utilisées. C'est la théorie du polymorphisme extensionnel, sujet de thèse de Jun Furuse. Cette approche permet aussi de formaliser les calculs qui dépendent dynamiquement des types comme les entrées-sorties ou les valeurs dynamiques.

3.1.4 Typage et confidentialité

De nombreux programmes manipulent des données privées pour des raisons commerciales ou personnelles. Comment empêcher la diffusion inopinée de ces données confidentielles à des tiers potentiellement mal intentionnés ?

Depuis plusieurs décennies, on cherche des moyens de contrôler la dissémination malencontreuse d'information au cours du déroulement des programmes. Une solution prometteuse consiste à analyser les programmes, de façon à prouver avant toute exécution, qu'ils se comporteront de façon acceptable. Ce procédé est généralement nommé analyse de *flots d'information*.

Là encore, l'utilisation de systèmes de types est féconde : François Pottier et Sylvain Conchon ont défini un système de types pour l'analyse de flots d'information dans un langage purement fonctionnel, ce qui permet d'appliquer à l'analyse de flots d'information toutes les techniques de typage classiques : polymorphisme, sous-typage. François Pottier et Vincent Simonet travaillent actuellement à la poursuite de ce travail, en cherchant à étendre ces résultats à des langages de programmation plus réalistes.

3.2 Le langage Caml

Glossaire :

Styles de programmation et structuration des programmes Les programmes s'écrivent dans des langages de programmation, et peuvent être organisés et structurés selon des styles différents.

Le style **fonctionnel** tend à privilégier l'usage des fonctions, presqu'au sens mathématique du terme : un programme est une expression à évaluer qui fournit un résultat qui ne dépend que des valeurs des paramètres d'entrée.

Plus classique, le style **impératif** privilégie l'exécution d'instructions qui modifient l'état de la machine (sa mémoire ou bien ses périphériques d'entrées-sorties). Le résultat du programme est alors tout ou partie de l'état de la machine à la fin de l'exécution.

Le style **à objets** se base, lui, sur des entités qu'on appelle objets. Ces objets regroupent des données, leur état (sorte de mémoire privée aux données) et le comportement de ces données (qu'on appelle *méthodes*). Afin de favoriser la réutilisation de code, la définition d'objets complexes utilise généralement un mécanisme qui leur permet d'hériter des fonctionnalités d'objets plus simples. Ce mécanisme s'appelle l'*héritage* : à la définition d'un objet seules les fonctionnalités nouvelles et spécifiques doivent être programmées, les autres sont « récupérées » d'objets parents déjà définis. La programmation par objets rencontre un grand succès dans l'industrie du logiciel.

Le développement d'applications de grande taille nécessite aussi la division des programmes en unités de taille raisonnable, à mi-chemin entre la granularité fine des fonctions ou des objets, et celle d'un gros programme monolithique : il s'agit des **modules**, qui regroupent des sous-programmes appartenant à une même entité logique.

Compilation et exécution Le compilateur d'un langage de programmation traduit les programmes écrits dans ce langage en des instructions de plus bas niveau, interprétables par la machine. On distingue les instructions de *code par octets* (ou *bytecode*) et les instructions de *code natif*. Les instructions de bytecode sont indépendantes de la machine qui exécute le programme : elles définissent une véritable **machine virtuelle** qui est réalisée par un programme dédié qui exécute les instructions de bytecode, l'*interprète de bytecode*. Le même jeu d'instructions de bytecode est donc utilisable pour une grande variété d'architectures de machines : toutes les architectures pour lesquelles on dispose de l'interprète de bytecode. En revanche, les instructions de **code natif** sont spécifiques à l'architecture matérielle de la machine d'exécution : le jeu d'instructions est celui de la machine d'exécution.

Ces deux types de code présentent des avantages et des inconvénients différents : s'il produit du bytecode, le compilateur du langage de haut niveau est simplifié puisqu'il ne doit produire qu'un seul type d'instructions pour tous les types de machines. S'il produit du code natif, le travail du compilateur est bien plus difficile puisque le code produit change pour chaque type de machine, mais en revanche le code obtenu est plus rapide à l'exécution puisque les instructions du code natif sont directement interprétées par la machine.

Le **modèle d'exécution** du langage est une vision abstraite de l'exécution du code produit par le compilateur et de sa politique de gestion de la mémoire.

Résumé : *Le langage de programmation Caml est l'un des langages de la famille ML. Comme toutes les autres versions de ML, Caml a été à la fois la source d'ins-*

piration et l'objet de nombreuses recherches, et s'est souvent enrichi des solutions apportées aux problèmes de recherche, tant dans le domaine du typage que dans celui des modèles d'exécution.

Autorisant les styles de programmation impératif, fonctionnel et par objets, dotée d'un puissant système de modules et d'un compilateur très performant, la dernière version de Caml développée au projet Cristal s'appelle Objective Caml (OCaml) et constitue une base de développement d'applications réelles dans les domaines les plus divers.

Parmi les développements du projet Cristal, le langage Caml[WL99] joue un rôle important. L'équipe en tire en effet des sujets de recherche et l'utilise non seulement comme champ d'expérimentation et de validation, mais aussi comme moyen privilégié de transfert et de diffusion des résultats.

Le langage Caml est l'héritier des premières versions de ML, le méta-langage de l'assistant de preuves LCF conçu par Robin Milner. ML servait alors à programmer les tactiques de recherche de preuves du système LCF.

Constitué initialement d'un noyau fonctionnel et de quelques traits impératifs, le langage ML connut des évolutions majeures au cours des années, tant du point de vue de ses caractéristiques que de l'efficacité de ses implémentations. Deux branches distinctes de ML sont apparues au milieu des années 80 : Standard ML et Caml. Standard ML est le résultat du travail de conception et de définition de ML effectué par un groupe placé sous la direction de Robin Milner. Un système de modules original formait le trait le plus saillant et novateur de Standard ML. Le langage Caml constitue l'autre branche de cette famille de langages : il a été conçu et développé au sein du projet Formel à l'INRIA, en collaboration avec des membres du Laboratoire Informatique de l'École Normale Supérieure de Paris, puis au projet Cristal. Caml a lui aussi connu d'importantes évolutions.

Les recherches menées au projet Cristal ont conduit à l'adjonction au langage Caml de traits importants comme les modules et les objets. En effet, ces deux façons de structurer les programmes ont été l'objet de recherches intensives afin de les doter de sémantiques statiques (typage) et dynamiques (exécution) clairement définies et prouvées correctes. Les objets ont été étudiés par Didier Rémy (cf. section 3.1.3). Les modules ont été étudiés par Xavier Leroy[Ler96], qui a étendu et simplifié le système de modules de Standard ML pour le rendre compatible avec la compilation séparée. (Modularité et compilation séparée sont deux éléments essentiels du développement logiciel à grande échelle.)

La version de Caml qui incorpore tous ces traits s'appelle Objective Caml.

Du point de vue de l'implémentation, le langage Caml a aussi donné lieu à des recherches sur les modèles d'exécution des langages fonctionnels. Le modèle d'exécution de Caml était initialement basé sur la Machine Abstraite Catégorique (CAM) et s'appuyait sur la machine LLM3 de Le-Lisp (de l'INRIA). Il a été changé en une machine virtuelle bytecodée avec l'implémentation Caml-Light[LW93]. Xavier Leroy a réalisé cette implémentation conçue pour être

[WL99] P. Weis, X. Leroy, *Le langage Caml*, Dunod, juillet 1999, Deuxième édition.

[Ler96] X. Leroy, « A syntactic theory of type generativity and sharing », *Journal of Functional Programming 6*, 5, 1996, p. 667–698.

[LW93] X. Leroy, P. Weis, *Manuel de référence du langage Caml*, InterÉditions, juillet 1993.

portable et économe en ressources mémoire, et qui s'appuie sur le gestionnaire de mémoire réalisé par Damien Doligez[DL93]. Lors du passage de Caml Light à Objective Caml, Xavier Leroy a ajouté au générateur de bytecode un tout nouveau compilateur qui produit du code natif performant. L'alliance de ces deux compilateurs permet dorénavant de disposer du meilleur des deux mondes : portabilité et cycle de développement réduit grâce au bytecode, performance et efficacité grâce au code natif que produit le compilateur optimisant (qui est disponible pour les architectures les plus courantes).

4 Domaines d'applications

4.1 Sécurité de programmation et rapidité du développement

Un des objectifs du typage est la détection rapide d'erreurs de programmation. Les recherches dans ce domaine sont donc de portée très générale, puisqu'elles ont potentiellement pour objet et pour application tout le champ de la programmation. Lorsqu'on allie à cette détection d'erreurs des facilités de structuration du code (fonctions, modules, objets), on améliore l'abstraction et la réutilisabilité et l'on se donne les moyens de maîtriser l'évolution des systèmes complexes.

De fait, plusieurs expériences ont montré que les développements réalisés dans un langage de programmation comme Objective Caml augmentent de façon considérable la productivité du développement, et facilitent grandement la maintenance. Objective Caml, même s'il trouve ses origines dans le domaine du calcul symbolique, a été utilisé avec succès dans des contextes divers : gestion et maintenance d'équipements téléphoniques, applications réparties dans le domaine du travail coopératif, compilateurs, outils d'accès au Web, *etc.*

4.2 Programmation d'applications de haute sécurité

Les méthodes utilisées dans l'étude des systèmes de types et la preuve de leurs propriétés sont aussi applicables à la spécification et la vérification de politiques de sécurité. C'est un domaine actuellement très actif à cause du succès du code mobile et des cartes à puce « ouvertes » (cartes Java), car il s'agit de programmes qui s'exécutent dans un contexte très particulier, et pour lesquels il semble possible de définir des politiques de sécurité raisonnables. Nous participons activement à cette voie de recherche.

4.3 Interopérabilité

L'utilisation d'un nouveau langage de programmation comme Objective Caml dans des applications réalistes ou industrielles nécessite souvent l'utilisation de bibliothèques ou de composants logiciels développés dans d'autres langages. Symétriquement, on peut souhaiter écrire en Objective Caml les parties les plus algorithmiquement difficiles d'une application alors

[DL93] 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, p. 113–123, 1993.

que les autres parties (interface utilisateur, harnais de communication, programme principal) sont déjà écrites dans un langage imposé.

C'est pourquoi l'interopérabilité entre Objective Caml et d'autres langages, soit directement, soit via une architecture standard de composants logiciels (Corba, COM, COM+) est l'une de nos priorités.

4.4 Enseignement de la programmation

Nos travaux en conception et mise en œuvre de langages de programmation ont une retombée importante dans le domaine de l'enseignement. Caml-Light est en effet l'un des langages recommandés pour l'enseignement de l'option Informatique dans les classes préparatoires. Caml-Light et OCaml sont aussi largement utilisés dans les écoles d'ingénieurs et les universités, en France et à l'étranger.

4.5 Linguistique computationnelle

Le domaine de la « Linguistique computationnelle » ou « Traitement automatique des langues naturelles » consiste à utiliser les moyens de calcul des ordinateurs pour traiter les problèmes linguistiques des langues naturelles. Ce domaine semble être maintenant arrivé à une certaine maturité vis-à-vis des applications (notamment en ce qui concerne la recherche d'informations dans des bases textuelles comme le Web). En outre le traitement automatique des langues naturelles combine plusieurs problématiques dont certaines recouvrent fortement des thèmes où l'INRIA a de fortes compétences (théorie des types, analyse syntaxique, logique computationnelle), et il semble donc prometteur de susciter un certain renforcement de notre implication dans cette discipline en plein essor.

Gérard Huet a commencé cette année une réflexion prospective approfondie sur l'état de l'art de ce domaine, en France, en Europe, et plus généralement dans le monde. Une première constatation est que des progrès significatifs ne peuvent être espérés dans ce domaine multidisciplinaire sans une implication sur un large spectre de problématiques (phonétique, morphologique, lexicale, grammaticale, syntaxique, logique, sémantique, pragmatique) faisant appel à de nombreuses technologies (transductions, compilation, programmation logique, représentation des connaissances, bases de données de corpus, analyse statistique) et qu'une coopération large et notamment multilingue s'impose.

Ce nouvel axe de recherche fait donc appel à une large panoplie d'outils théoriques et pratiques de notre domaine : compilation, analyse syntaxique et sémantique, typage. L'écriture de maquettes de programmes typiques du domaine (construction d'un dictionnaire, analyse de mots fléchis, allitération phonétique) se solde par un bilan extrêmement positif et encourageant pour le projet Cristal puisque l'ensemble d'outils OCaml et Camlp4 apparaît bien adapté au développement d'une plate-forme modulaire fiable et efficace pour le traitement des langues naturelles.

À terme, ce nouvel axe de recherche pourrait donc permettre au projet Cristal de mettre en lumière ses points forts, tant théoriques que pratiques, puisque nos techniques d'analyse formelle des langages (tant analyse syntaxique que typage ou autres analyses statiques) paraissent pertinents pour cette discipline et que nos produits logiciels sont bien adaptés à la

fabrication des outils informatiques correspondants.

5 Logiciels

5.1 Implantations de Caml

Les systèmes Caml (Objective Caml et Caml-Light) développés au sein du projet Cristal sont des logiciels libres distribués sur le réseau et présents sur un certain nombre de CD-ROMs gratuits ou vendus à prix coûtant.

Objective Caml est un langage de programmation généraliste. Autorisant les styles de programmation impératif, fonctionnel et par objets, doté d'un puissant système de modules et d'un compilateur très performant, Objective Caml permet le développement et la maintenance d'applications complexes et efficaces.

Caml-Light en est une version allégée, plus particulièrement destiné à être utilisé comme support de l'enseignement de la programmation.

Les distributions et documentations de ces logiciels sont disponibles à l'adresse `http://caml.inria.fr/`.

5.2 Outils syntaxiques

Les travaux de Michel Mauny et Daniel de Rauglaudre sur l'intégration en ML d'outils de manipulation de syntaxes concrètes (analyse syntaxique fonctionnelle, quotations, grammaires extensibles, etc.) se sont concrétisés en 1996 par la mise en œuvre par Daniel de Rauglaudre d'un préprocesseur du langage Caml appelé Camlp4. Camlp4 permet à l'utilisateur de substituer son propre analyseur syntaxique à celui d'Objective Caml, et donc d'étendre le langage ou même de le restreindre et de le spécialiser à telle ou telle application.

6 Résultats nouveaux

6.1 Systèmes de types

6.1.1 Extension de ML avec des types de second ordre

Participants : Didier Rémy, Didier Le Botlan.

Le langage ML manipule des types de premier ordre (sans quantificateurs), enrichis par des schéma de types qui autorisent la quantification universelle, mais seulement de façon externe. Cela permet de synthétiser les types très simplement, par un calcul d'unification de premier ordre, mais limite leur expressivité. Symétriquement, le système F autorise des types de second ordre (les quantificateurs peuvent être imbriqués à une profondeur arbitraire) ce qui lui confère une plus grande expressivité, mais rend indécidable la synthèse de types.

Didier Le Botlan et Didier Rémy ont étudié un système mixte ML^F qui combine la synthèse des types de ML et l'expressivité du système F. Ce travail généralise une proposition antérieure

de Didier Rémy et Jacques Garrigue[GR99], qui permettait déjà d'encapsuler des types de second ordre dans un système de premier ordre, mais en maintenant une différence entre les schémas de types inférés et les types explicites de second ordre indiqués par l'utilisateur.

Cette différence disparaît dans ML^F, grâce à un enrichissement des types de second ordre avec la possibilité de quantifier sur l'ensemble de tous les types (comme dans le système F) mais aussi sur l'ensemble de toutes les instances d'un type polymorphe ou encore sur un seul type : cette dernière forme est utilisée pour exprimer le partage au sein des types, conservé au cours du typage, qui joue une rôle essentiel pour distinguer les informations synthétisées de celles données explicitement.

Le système obtenu conserve l'inférence complète des types pour tous les programmes ML tout en permettant un accès souple aux termes du système F par une simple annotation des paramètres de fonction utilisés de façon polymorphe.

La formalisation complète de ML^F, incluant un algorithme d'unification, un algorithme d'inférence de types et leurs preuves de correction est actuellement en cours. Un programme prototype permettant le typage des programmes a été également réalisé, permettant de valider aussi cette proposition par la pratique.

6.1.2 Objets et concurrence

Participants : Cédric Fournet [Microsoft Research, Cambridge], Cosimo Laneve [université de Bologne], Luc Maranget [projet Moscova], Didier Rémy.

Didier Rémy a finalisé ses travaux sur l'adjonction au calcul *Join* d'objets et de classes primitifs. Ce calcul repose d'abord sur un transfert de la technologie des langages à objets et à classes et de leur typage, telle que celle développée pour le langage Ocaml, à un langage concurrent, le calcul *Join*.

Cependant le contexte concurrent requiert de nouvelles constructions qui n'ont pas de contrepartie dans le cas séquentiel. En particulier, nous avons fourni une construction d'héritage appelée « raffinement sélectif » qui permet d'hériter des clauses des classes parentes tout en modifiant leur synchronisation, et ceci de façon sélective en utilisant un mécanisme de filtrage sur leur motif de synchronisation dans la classe parente. Cette construction très expressive permet de résoudre élégamment des problèmes difficiles souvent mentionnés dans la littérature sous le nom d'« anomalies d'héritage ».

Une partie[FMLR00] de ce travail, décrivant notamment l'expressivité du langage a été présentée à la conférence *Foundations of Software Technology and Theoretical Computer Science* (FSTTCS) en décembre 2000 à New Delhi (Inde). Une version détaillée a été soumise à publication.

[GR99] J. Garrigue, D. Rémy, « Extending ML with Semi-Explicit Higher-Order Polymorphism », *Journal of Functional Programming 155*, 1/2, 1999, p. 134–169, une version préliminaire a été présentée à TACS'97.

[FMLR00] C. Fournet, L. Maranget, C. Laneve, D. Rémy, « Inheritance in the Join Calculus », *in: Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science, 1974*, Springer, décembre 2000, `http://cristal.inria.fr/~remy/work/ojoin/`.

6.1.3 Types polymorphes contraints

Participant : Daniel Bonniot.

Dans le cadre de sa thèse, Daniel Bonniot a cherché à améliorer les systèmes de types pour langages à multi-méthodes. Il a pour ce faire développé une présentation stratifiée des systèmes de types. Ces travaux sont en cours de soumission [18].

Il a également commencé à inclure dans ce cadre un système de types avec contraintes sur des sortes. Ces types sont utiles pour typer des opérations présentant du polymorphisme *partiel* de sous-typage, telles que les opérateurs arithmétiques. Ainsi, l'opération $x \mapsto \frac{1}{x}$ qui a en particulier les types monomorphes $numeric \mapsto numeric$, $rational \mapsto rational$ et $int \mapsto rational$ avec $num \geq rational \geq int$.

Enfin, Daniel Bonniot a étudié la modélisation de la construction `null` (représentant un pointeur non initialisé) uprésente dans la plupart des languages impératifs, mais qui compromet la sûreté du typage. L'approche ML classique consistant à introduire un type paramétré *option* est moins puissante en ce qu'elle ne fournit pas le sous-typage $t \leq t\ option$. Ce sous-typage est sans doute désirable car il correspond à l'intuition : si une variable contient un entier, on peut considérer qu'elle contient soit un entier soit `null`. Il est utilisé implicitement dans les language impératifs, si l'on considère que tous les types sont des types option (puisqu'ils ont `null` comme valeur possible). Pour rendre le typage sûr, il faut rendre apparente la distinction entre t et $t\ option$. Cette étude est particulièrement intéressante dans le cadre des multi-méthodes car il y a interaction avec le test de couverture. Si une méthode accepte des arguments nuls, elle doit aussi pourvoir les implémentations correspondantes. On voit aussi apparaître la possibilité de *dispatcher* sur la valeur `null`, ce qui permet des définitions de méthodes plus claires : on remplace un test conditionnel par un dispatch. Ainsi la longueur d'une liste chaînée

```
length(x) = if (x == null) then 0 else 1 + length(x.tail)
```

devient similaire à la présentation avec filtrage

```
length(null) = 0
length(x) = 1 + length(x.tail)
```

mais tout en conservant l'avantage des multi-méthodes de permettre l'extension par de nouveaux cas.

6.1.4 Sous-typage atomique

Participants : François Pottier, Vincent Simonet.

Dans le but de réaliser une implantation réaliste du système de types pour l'analyse de flots d'information présenté dans [10], Vincent Simonet s'est intéressé au problème de l'inférence de types dans un système à base de contraintes dans le cas du sous-typage atomique. Cela nécessite la mise au point d'algorithmes permettant d'une part de tester la satisfiabilité d'ensemble de contraintes et d'autre part de simplifier ces contraintes (pour des raisons d'efficacité mais aussi de lisibilité des schémas de type affichés à l'utilisateur). La présentation formelle de ces algorithmes, accompagnée de leur preuve, ainsi que leur implantation en Objective Caml ont été menées de front.

6.1.5 Isomorphismes de types avec variants

Participants : Roberto Di Cosmo, Vincent Balat [PPS, Univ. Paris 7], Marcelo Fiore [Cambridge Univ, UK].

L'étude des isomorphismes de types est l'étude des types de données qui sont codifiables l'un dans l'autre sans perte d'information. L'idée de base est très simple : une donnée a de type A peut être « codée » par un programme (ou fonction) f sous la forme d'une donnée $f(a)$ de type B sans perte d'information s'il est possible de « décoder » ensuite $f(a)$ à l'aide d'un programme (ou fonction) g tel que $g(f(a))$ soit a lui-même. Un exemple simple de tels isomorphismes et très connu dans la programmation fonctionnelle est le suivant :

$$A \to (B \to C) \cong (A \times B) \to C$$

Cette étude des isomorphismes a des applications pratiques immédiates dans la conception d'outils de recherche de fonctions dans des bibliothèques de sous-programmes (dans les langages de programmation) ou de théorèmes (dans les assistants de preuve). De tels outils ont été effectivement réalisés les années passées en collaboration avec David Delahaye, Xavier Leroy, Jerome Vouillon, Pierre Weis et Benjamin Werner dans des versions récentes de Caml-Light et de Coq.

Jusque là, une remarquable correspondance entre isomorphisme de types, fonctionnelles inversibles, objets isomorphes dans les catégories cartésiennes fermées et identités arithmétiques sur les entiers (un cas particulier du « *Tarski's High School Algebra problem* »), permettait une approche unifiée et élégante, quoique complexe, du problème.

En coopération avec Marcelo Fiore, de l'Université de Cambridge (UK), Vincent Balat et Roberto Di Cosmo étudient les isomorphismes de types en présence de sommes (types *variants*) : il s'agit d'un problème bien plus complexe que les précédents, mais des résultats préliminaires montrent de façon surprenante que les types variants sont radicalement différents des sommes dans les modèles de l'arithmétique. Puisque les identités arithmétiques avec sommes ne sont pas finiment axiomatisables, ce résultat de séparation nous redonnent l'espoir de pouvoir capturer les isomorphismes de type par une théorie équationnelle finie.

Un article rendant compte de ces recherches est en cours de rédaction.

6.1.6 Polymorphisme extensionnel et fonctions génériques

Participants : Pierre Weis, Jun Furuse.

Jun Furuse a principalement consacré l'année 2001 à la rédaction de sa thèse. Cependant, en collaboration avec Pierre Weis, il a continué les travaux théoriques sur ce système de types, en particulier les méthodes de compilation efficace de la sélection des clauses des fonctions génériques et leur prédiction statique ce qui simplifie et étend la sémantique du filtrage des clauses des fonctions génériques. Jun Furuse et Pierre Weis ont aussi étudié l'ajout de cas supplémentaires à l'ensemble des clauses d'une fonction générique, l'*extension* de la fonction générique. La sémantique adoptée est celle de l'inclusion (statique) des cas de filtrage, une méthode tout-à-fait analogue à l'extension des modules d'Objective Caml.

L'étude de l'inclusion de valeurs polymorphes dans les *dynamiques* (ou valeur du type `dyn`) a été poursuivie. La méthode de typage proposée est simple et cohérente avec le reste du langage : elle consiste à typer `dyn` e comme une définition ordinaire ou e serait l'expression définissante. Les mêmes restrictions de typage que dans le cas de la construction `let` sont appliquées. Enfin une phase finale de vérification consiste à unifier au type prédéfini `dyn` toutes les variables de type libres et non généralisées qui apparaissent dans le type de e.

Une autre idée prometteuse apparue cette année consiste à profiter des contraintes de type additionnelles présentes dans les schémas de type des fonctions génériques pour supprimer la distinction syntaxique entre variables de type dynamiques et variables de type statiques (les variables de type dynamiques sont notées actuellement `$a`, `$b`, ..., alors que les variables habituelles de Caml sont notées `'a`, `'b`, ...). Ainsi, l'utilisateur du langage n'aurait plus à connaître l'existence des variables dynamiques, ce qui simplifierait l'implémentation du formalisme et son acceptation par la communauté des utilisateurs de Caml.

Courant 2001, Jun Furuse a rendu publique *GCaml* (`http://cristal.inria.fr/~furuse/generics/index.html`), la maquette grandeur nature qui implémente le polymorphisme extensionnel. Basée sur un compilateur Objective Caml 2.02, cette distribution est pleinement fonctionnelle et a été testée par plusieurs utilisateurs. Les nouveaux traits implémentés ont fait l'objet de nombreux messages dans la *mailing-list* Caml, qui ont montré l'intérêt certain que les utilisateurs de Caml portent à l'extension proposée.

6.2 Modules et mixins

Participants : Xavier Leroy, Tom Hirschowitz.

La *modularisation* est le processus de division d'un programme en unités indépendantes, compréhensibles individuellement par le programmeur, qui spécifie leur assemblage. Un programme modulaire est plus facilement lisible : il est décomposé en des parties correspondant à l'intuition. La *compilation séparée* consiste à diviser un programme en unités compilables indépendamment les unes des autres par le compilateur. Un avantage de la compilation séparée est qu'elle facilite grandement la réutilisation du code.

Un langage de programmation se doit de fournir des outils pour les deux processus. Le système OCaml permet la compilation séparée et possède un système de modules puissants. Cependant, sur les deux plans présentés, des progrès restent à faire.

- Du point de vue de la modularisation, l'intuition suggère parfois de diviser les programmes en fragments mutuellement dépendants. Cela n'est malheureusement pas possible dans le cadre des modules d'OCaml.
- Du point de vue de la réutilisation du code, il est impossible de reprendre un module compilé existant et d'en modifier une composante, ou bien d'en ajouter de nouvelles. Ces mécanismes, appelés *redéfinition* et *héritage* dans les langages orientés objets, seraient vraiment profitables au système de modules d'OCaml.

Dans les deux cas, des astuces existent pour contourner les problèmes, mais des outils mieux définis seraient bienvenus.

Dans ce contexte, des systèmes primitifs de modules *mixins* ont été formulés, qui permettent les dépendances mutuelles entre modules, l'héritage et la redéfinition. Toutefois, ils se placent

dans le contexte de langages en appel par nom, où toutes les formes de récursion sont autorisées. Malheureusement, en OCaml seules les valeurs peuvent être récursives : c'est typiquement le cas des fonctions. Tom Hirschowitz et Xavier Leroy ont mis au point un système de typage pour les modules mixins qui permet d'éviter les écueils de la récursion mal fondée. Ils ont de plus mis en place un schéma de compilation pour ce système, compatible avec la compilation séparée, vers le langage λ_B, un λ-calcul avec enregistrements et `let rec`. Néanmoins, ce schéma de compilation génère des termes habituellement rejetés par tous les systèmes de typage du λ-calcul. Tom Hirschowitz et Xavier Leroy ont donc mis au point un nouveau système de typage pour λ_B, qui accepte les programmes générés par le schéma de compilation, tout en garantissant leur sûreté. Les résultats de cette recherche sont un article soumis à publication [19] et sa version longue, avec les preuves de sûreté [20]. Un travail important reste à faire pour étendre le système au langage OCaml dans son ensemble, notamment pour ce qui est des types concrets définis par l'utilisateur.

6.3 Analyses statiques

6.3.1 Analyse de flots d'information

Participants : François Pottier, Vincent Simonet.

Comment protéger les données stockées dans un système informatique contre toute modification ou divulgation indésirable – d'origine accidentelle ou intentionnelle – tout en permettant l'accès à ces données à tout programme qui en fera une utilisation légitime ? Une solution prometteuse consiste à analyser préalablement chaque programme. Si l'analyse garantit que son comportement sera acceptable, on peut lui accorder un accès sans restriction. Dans le cas contraire, son exécution sera refusée. Cette technique, appelée analyse de *flots d'information*, présente l'intérêt de ne reposer sur aucune notion de confiance – seule la correction de l'analyse importe et peut être prouvée formellement.

L'an passé, François Pottier et Sylvain Conchon (projet Moscova) ont montré qu'un système d'analyse de flots d'information pour un langage purement fonctionnel peut être dérivé, de façon systématique, à partir d'un système de types standard, et ont fourni une preuve élémentaire de ce résultat. Cette année, Vincent Simonet et François Pottier ont généralisé ces résultats en présentant une analyse de flots d'information pour un langage fonctionnel doté de traits impératifs (i.e. de références) et d'exceptions, comparable au langage Caml-Light. La correction de cette analyse a été établie, ce qui constitue un résultat nouveau. Un article décrivant ces travaux [10] sera présenté à la conférence *Principles of Programming Languages* (POPL) en janvier 2002.

6.3.2 Vérification de bytecode Java sur cartes à puces

Participant : Xavier Leroy.

L'architecture Java Card représente une évolution majeure dans le monde de la carte à puce : on passe de cartes mono-applications, programmées dans des langages de bas niveau (C, assembleur) et inévolutives après émission à des cartes multi-applications, programmées dans un langage sûr (Java) et permettant le téléchargement de nouvelles applications (applets)

après émission de la carte. Ces évolutions ne vont pas sans soulever de nouveaux problèmes de sécurité informatique, objets de nombreux travaux de recherche à l'INRIA et ailleurs.

En particulier, il est établi que le téléchargement d'applets Java n'est sûr que si les applets sont soumises lors du chargement à une vérification statique appelée vérification de bytecode. Or, ce composant crucial de l'architecture de sécurité Java n'était jusqu'à récemment infaisable sur une carte à puce, en raison de la complexité de cette vérification, qui dépasse les faibles ressources de calcul disponibles sur une telle carte.

Dans le cadre d'une convention de concours scientifique avec la société Trusted Logic, Xavier Leroy travaille à la conception et à l'implémentation de vérificateurs de bytecode Java suffisamment économes en ressources pour être embarqués sur une carte à puce. L'idée de base est que la partie embarquée du vérificateur peut être drastiquement simplifiée en tirant profit de transformations fines effectuées sur le code de l'applet en dehors de la carte, sans pour autant affaiblir la sécurité. Ces techniques sont décrites dans un article présenté au congrès E-Smart 2001 [9] et à paraître dans *Software : Practice and Experience* [2]. L'expertise sur la vérification de bytecode acquise lors de ce travail a valu à Xavier Leroy d'être invité à présenter un exposé d'introduction à cette problématique au congrès *Computer Aided Verification* [8].

6.3.3 Flots synchrones et programmation fonctionnelle

Participants : Michel Mauny, Pascal Cuoq.

Pascal Cuoq travaille en collaboration avec le Laboratoire d'Informatique de Paris 6, et son sujet de thèse a pour objectif général l'amélioration du langage Lucid Synchrone, développé au LIP6 par Marc Pouzet.

Lucid Synchrone est un langage de flots synchrones, appartenant à la même famille que Lustre. Lucid Synchrone introduit des notions venant de la programmation fonctionnelle typée, telles que des analyses statiques inférées (typage, horloges), la modularité, et l'ordre supérieur.

Cette année, Pascal Cuoq a implémenté l'analyse de causalité présentée dans [4] au sein du compilateur Lucid Synchrone. Cette implémentation a permis de mettre en évidence un certain nombre de cas qui sont rejetés par cette analyse, bien que correspondant à des programmes réels. Pascal Cuoq a donc travaillé avec Marc Pouzet à étendre l'expressivité de l'analyse de causalité de manière à accepter ces programmes. Les extensions de l'analyse de causalité ont été implémentées et testées, et la preuve de leur correction est l'un des objectifs de Pascal Cuoq en vue d'une publication.

Pascal Cuoq a implémenté un prototype d'interpréteur de flots, beaucoup plus léger et mieux intégré avec Objective Caml que le compilateur Lucid Synchrone. Cet interpréteur utilise Camlp4 pour étendre OCaml par des constructions synchrones.

6.4 Implémentations de Caml

6.4.1 Objective Caml

Participants : Xavier Leroy, Jacques Garrigue [université de Kyoto], Luc Maranget [projet Moscova], Daniel de Rauglaudre, Damien Doligez [projet Moscova], Pierre Weis,

Jérôme Vouillon [PPS, CNRS, Univ. Paris 7].

Objective Caml est notre implémentation la plus récente du langage Caml. Elle ajoute au langage Caml de base un système complet d'objets et de classes, mettant Caml au niveau des meilleurs langages orientés-objets existants ; un calcul de modules puissant, mais néanmoins compatible avec la compilation séparée ; et un compilateur produisant du code assembleur de hautes performances pour la plupart des processeurs du marché (Pentium, Alpha, PowerPC, Sparc, Mips, HPPA, StrongArm, IA64).

Cette année, nous avons rendu publiques les versions 3.01, 3.02 et 3.03 d'Objective Caml. Les principales innovations de ces versions sont :

- La révision des mécanismes d'arguments de fonctions nommés ou optionnels, introduits par Jacques Garrigue dans la version 3.00 en 2000. La réalisation initiale de ces mécanismes visait à la flexibilité maximale, mais posait des problèmes techniques vis-à-vis de l'inférence de types. La version révisée corrige le tir en imposant par typage statique un usage plus cohérent des noms d'arguments, tout en préservant la compatibilité arrière avec Objective Caml version 2.
- L'introduction du chargement dynamique de la partie C des bibliothèques mixtes Caml/C, en alternative au lien statique utilisé jusqu'ici. Ce mécanisme introduit une plus grande flexibilité dans l'utilisation de ces bibliothèques mixtes, et rend les exécutables *bytecode* liés avec ces bibliothèques indépendants de la machine.
- L'intégration plus étroite du pré-processeur Camlp4, qui fait maintenant partie de la distribution standard et se charge du traitement de certaines extensions syntaxiques (parseurs de flux) auparavant traitées dans Objective Caml lui-même. Cette intégration a été complétée par la possibilité de personnaliser toutes les sorties de la boucle interactive.
- L'amélioration de la compilation du filtrage (définitions par cas), ainsi que de la rapidité de compilation via l'utilisation d'algorithmes plus efficaces pour certaines phases du typeur. Le travail concernant la compilation optimisée du filtrage est décrit plus en détails dans le rapport d'activités du projet Moscova.

6.4.2 Le préprocesseur Camlp4

Participant : Daniel de Rauglaudre.

Camlp4 est un préprocesseur pour OCaml, permettant de programmer des extensions syntaxiques dans OCaml, de changer la syntaxe du langage, de faire du formattage (*pretty print*) de programmes OCaml, et offrant un système de grammaires récursives descendantes dynamiquement extensibles. En plus de la syntaxe normale d'OCaml, Camlp4 propose une syntaxe dite « révisée », qui tente de résoudre les quelques problèmes de la syntaxe normale.

Daniel de Rauglaudre a ajouté un système de fonctions extensibles, définies par filtrage de motifs. La souplesse fournie par ce mécanisme est bien sûr obtenue au détriment de la rapidité. Les fonctions extensibles permettent néanmoins de compléter utilement les syntaxes extensibles, en autorisant le programmeur à étendre les fonctions de traitement d'arbres de syntaxe abstraite obtenus par les extensions de syntaxe concrète.

Il a également ajouté un système de flots fonctionnels avec analyseurs syntaxiques utilisant le rebroussement limité. Rebroussement parce que si une règle formée de deux ou plusieurs

symboles consécutifs échoue, la règle suivante est essayée (au lieu de faire « erreur de syntaxe » comme dans les flots normaux). Limité parce que dans ce cas, le système n'essaie pas les règles suivantes des symboles de la même règle qui pourraient offrir une autre combinaison.

Daniel de Rauglaudre a aussi apporté un certain nombre d'améliorations à Camlp4. Parmi les plus visibles, citons l'ajout de directives traitées par Camlp4 (et donc invisibles à OCaml) dans les fichiers source dans le but d'améliorer la commodité d'utilisation, l'ajout de facilités d'interfaçage entre les analyseurs lexicaux produits par `ocamllex` et les analyseurs de flots, et un traducteur de grammaires (qui ne doivent plus être étendues) en fonctions d'analyse, ce qui a permis de gagner un facteur 3 sur la mise en œuvre habituelle des grammaires.

Enfin, cette année, Daniel de Rauglaudre a intégré les sources de Camlp4 dans celles d'OCaml. Cette intégration était devenue nécessaire à cause des dépendances fortes entre les deux systèmes. Cela a par ailleurs permis de reporter sur Camlp4 le traitement des flux et de ses analyseurs, et ainsi d'améliorer ce traitement.

Daniel de Rauglaudre a aussi écrit une documentation tutorielle [14] de Camlp4, moins ardue que le manuel de référence [13].

6.4.3 Interopérabilité

Participant : Xavier Leroy.

L'utilisation d'un nouveau langage de programmation tel qu'Objective Caml dans des applications réalistes ou industrielles nécessite souvent l'intégration avec des bibliothèques ou des composants logiciels développés dans d'autres langages. Par exemple, il est souvent préférable de réutiliser de grosses bibliothèques déjà écrites en C, C++, Fortran ou Java, plutôt que de les réécrire en Objective Caml. Symétriquement, on peut souhaiter écrire en Objective Caml les parties les plus algorithmiquement difficiles d'une application, les autres parties (interface utilisateur, harnais de communication, programme principal) étant déjà écrites dans un langage imposé.

Xavier Leroy a poursuivi le développement du générateur de code d'interface CamlIDL, dont la version 1.02 a été publiée en juillet 2001. À partir d'une description de haut niveau d'une interface exprimée dans le langage IDL (*Interface Description Language*), CamlIDL automatise la production du code d'interface (*stub code*) nécessaire à l'utilisation depuis Caml de fonctions écrites en C ou en tout autre langage disposant d'une interface externe avec C.

Parallèlement, Xavier Leroy continue son travail sur l'interfaçage entre Caml et Java. Cette interface met en correspondance les classes et objets Java avec des classes et des objets Caml. Elle s'appuie sur les mécanismes d'interfaces externes de Caml et de Java, en particulier pour « coupler » les gestionnaires mémoires de ces deux langages. Une version préliminaire de cette interface, de bas niveau et faiblement typée, a été rendue publique en juin 2001. Le travail se poursuit sur une interface de plus haut niveau et sûre vis-à-vis des types. Ceci pose d'intéressants problèmes de correspondance entre les systèmes de types et de classes des deux langages, qui sont significativement différents, et nécessite des extensions en cours d'étude du typage des objets en Objective Caml (typage polymorphe des méthodes).

6.4.4 Bibliothèques

OCamlODBC **Participant** : Maxence Guesdon.

OCamlODBC est une bibliothèque d'interface entre le langage OCaml et différents drivers ODBC. ODBC est un standard d'accès aux bases de donnés et est supporté par la plupart des systèmes de gestion de bases de donnés existants. Cette bibliothèque permet donc de développer des applications OCaml pouvant se connecter aux bases de données supportant ODBC.

Maxence Guesdon a poursuivi le développement de cette bibliothèque qu'il avait commencé avant de rejoindre le projet Cristal. De la documentation, des exemples, le support du *multi-threading* ont été ajoutés, ainsi qu'un script de configuration automatique facilitant la compilation.

OCamlODBC est disponible depuis `http://caml.inria.fr/devtools/ocamlodbc/ocamlodbc.html`.

Maintenance de bibliothèques existantes **Participants** : Jun Furuse, Pierre Weis.

En collaboration avec Pierre Weis, Jun Furuse a continué à maintenir et tenir à jour les bibliothèques Camlimages, CamlTk et MMM.

6.4.5 Outils de développement

Participant : Maxence Guesdon.

Aide à la documentation Maxence Guesdon a développé OCamldoc, un outil permettant de générer de la documentation à partir de code source OCaml commenté. Cet outil répond à une réelle demande d'un outil de documentation standard de la part des utilisateurs du langage OCaml, comme Javadoc l'est pour le langage Java.

OCamldoc est basé sur le même principe que Javadoc : des balises dans les commentaires du fichier source permettent de mettre en forme les renseignements associés aux fonctions, modules, classes, *etc.* Ocamldoc permet la génération de plusieurs formats de documentation dont LaTeX et HTML.

OCamldoc existe aussi sous forme de bibliothèque donnant accès aux fonctionnalités de récupération d'informations, ceci afin de pouvoir utiliser ces fonctionnalités depuis une autre application, comme le fait Cameleon.

Encore en développement, OCamldoc sera à terme inclus dans la distribution du langage OCaml. Il a d'ores et déjà été accueilli avec enthousiasme par les utilisateurs du langage. OCamldoc est disponible sur `http://caml.inria.fr/devtools/index.html`.

Aide à l'utilisation de bases de données depuis OCaml Maxence Guesdon a développé DBForge, un outil permettant de générer du code OCaml d'accès aux tables d'une base de données.

L'idée de cet outil vient du constat suivant : à partir de la description des tables d'une base de données, il est possible de générer *automatiquement* le code OCaml d'accès individuel à chacune de ces tables, c'est-à-dire le code des fonctions OCaml permettant d'effectuer les opérations `create`, `drop`, `select`, `insert`, `update` et `delete`.

La génération de code par DBForge présente de nombreux avantages par rapport à la génération manuelle de ce code : gain de temps, moins d'erreurs à l'exécution, et maintenance facilitée.

La génération des codes d'accès aux tables se fait ou bien via une interface textuelle ou bien une interface graphique.

DBForge ne génère pour l'instant que du code accédant à une seule table à la fois, et s'appuyant sur une interface comme OCamlODBC (cf. 6.4.4). DBForge a été utilisé pour développer le Hump (cf. 9.1.3).

DBForge existe aussi sous la forme d'une bibliothèque permettant à d'autres applications d'utiliser l'interface graphique de DBForge pour éditer des descriptions de bases de données, comme c'est le cas pour Cameleon.

DBForge est disponible sur `http://caml.inria.fr/devtools/index.html`.

Gestion de configuration via CVS Maxence Guesdon a développé OCamlCVS, une interface graphique permettant d'effectuer les commandes CVS courantes comme `commit` ou `update`, et d'afficher graphiquement les différences entre deux versions d'un fichier. OCamlCVS offre également une assistance à la résolution de conflits entre versions.

OCamlCVS existe aussi sous la forme d'une bibliothèque permettant d'intégrer facilement dans une application la gestion de fichiers gérés par CVS, en donnant accès aux fonctions courantes de CVS mais aussi à des composants d'interface graphique dont le comportement est paramétrable par l'application qui les utilise. Cameleon utilise cette bibliothèque pour gérer les fichiers sous CVS.

OCamlCVS est disponible sur `http://caml.inria.fr/devtools/index.html`.

Environnement de développement OCaml Maxence Guesdon a commencé le développement de Cameleon, un environnement de développement pour le langage OCaml. Muni d'une interface graphique, Cameleon offre les fonctionnalités suivantes :

- gestion de configuration, en utilisant la bibliothèque OCamlCVS (cf. 6.4.5) ;
- accès à la documentation : en utilisant la bibliothèque fournie avec OCamldoc, un navigateur de documentation intégré à Cameleon permet un accès rapide à la documentation issue de code source OCaml, ainsi que l'accès direct au code source depuis la documentation (on peut par exemple, à partir de la documentation d'une fonction, ouvrir le fichier source dans Efuns[1] ou Emacs à la ligne exacte de la définition de cette fonction) ;
- édition de fichier : Cameleon peut s'interfacer avec Emacs ou Efuns pour l'édition de fichiers texte, ou utiliser la bibliothèque DBForge (cf. 6.4.5) pour éditer des schémas de bases de données ;
- création de nouveaux fichiers à partir de modèles paramétrables par l'utilisateur ;
- définition de *types de fichiers* et association de propriétés à ces types de fichiers (éditeur, couleur, *etc.*).

Cameleon est disponible sur `http://caml.inria.fr/devtools/index.html`.

[1] Efuns est un clone d'Emacs développé en OCaml par Fabrice Le Fessant (projets Moscova/Sor).

6.5 Applications de Caml

6.5.1 Calcul parallèle

Participants : Roberto Di Cosmo, Xavier Leroy.

Dans le cadre de l'ACI GRID « CARAML », nous nous attaquons au développement de bibliothèques pour le calcul haute-performance et globalisé en OCaml. L'action réunit des chercheurs impliqués dans le parallélisme de données (laboratoires LIFO à Orléans, PPS à Paris 7, LACL à Paris 12), dans les calculs concurrents (projet Moscova), ainsi qu'une expertise en matière de typage et du langage OCaml au travers du projet Cristal. L'action s'appuie aussi sur l'activité OCamlP3L dans laquelle Roberto Di Cosmo et Xavier Leroy ont été impliqués en 1998, en collaboration avec l'Université de Pise.

6.5.2 Implémentation d'un langage de scripts en Caml

Participant : Bruno Verlyck.

Bruno Verlyck a collaboré avec le projet Cristal à raison d'une journée par semaine jusqu'à mi-juillet, de 4 journées par semaine ensuite, pour développer une bibliothèque permettant l'écriture de scripts en Caml. Cette bibliothèque, nommée **Cash** pour **Ca**ml **sh**ell, est inspirée du travail d'Olin Shivers sur Scsh (*a Unix Scheme shell*) (voir par exemple `ftp://ftp-swiss.ai.mit.edu/pub/su/scsh/scsh-manual.ps`).

Cash se compose d'une extension du module OCaml `Unix` et d'un sous-langage de composition de processus avec redirection et connexion des *file descriptors* permettant d'écrire des *pipelines* au sens *shell* avec une grande concision.

Une première distribution publique est envisagée pour le début 2002.

6.5.3 Logiciel Active Dvi

Participants : Pierre Weis, Jun Furuse, Didier Rémy, Xavier Leroy, Alan Schmitt [projet Moscova], Didier Le Botlan, Roberto Di Cosmo.

Active Dvi est un logiciel qui fait suite à Mldvi : un interprète de fichiers au format DVI utilisé principalement par TEX et LATEX. Cet interprète, réalisé par Alexandre Miquel (projet Logical) est réalisé en OCaml. Active Dvi est une extension de Mldvi qui permet l'inclusions de sons, d'animations, et plus généralement l'exécution de programmes arbitraires, dont les entrées-sorties se font *via* la page affichée. L'objectif est d'en faire un outil de présentation de transparents réalisés en LATEX, qui permette à la fois de disposer d'effets sophistiqués tout en restant bien adapté aux présentations scientifiques.

Les principales contributions à Active Dvi ont été cette année l'écriture d'un ensemble de macros LATEX, la reconception de l'architecture et l'écriture de nouveaux modules (Jun Furuse), l'adjonction d'animations (de texte, par exemple) et de transitions entre transparents (Jun Furuse), l'adjonction d'un mécanisme d'exécution de programmes externes (Pierre Weis et Jun Furuse), la couleur et les images de fond (Roberto Di Cosmo), l'interfaçage d'Active Dvi avec GhostScript et HyperRef (Didier Rémy), le copier-coller (Didier Le Botlan), le retaillage de la fenêtre (Alan Schmitt) et l'inclusion de dessins réalisés en PIC (Xavier Leroy).

La distribution d'Active Dvi est disponible avec le « bazar O'Caml » et comprend des fichiers d'exemples et de démonstration.

6.5.4 Serveur Web de bases de données généalogiques

Participant : Daniel de Rauglaudre.

GeneWeb est un logiciel de généalogie écrit en OCaml et qui fonctionne comme un service Web. Cette année, Daniel de Rauglaudre a distribué sept nouvelles versions du logiciel : des corrections de défauts, un peu de documentation, de très nombreuses extensions et améliorations. Les principales sont décrites ci-dessous.

L'aspect multilingue de Geneweb a été amélioré par l'ajout de 4 nouvelles langues, et par un traitement différentiel des variantes de la même langues. Daniel de Rauglaudre a amélioré l'affichage des arbres de parenté afin d'en permettre l'impression papier sur plusieurs pages, et de l'adapter à des navigateurs comme Lynx, utilisés par les mal-voyants. L'affichage des pages est maintenant personnalisable, la gestion de généalogies de grande taille a été optimisée.

Daniel de Rauglaudre a porté GeneWeb sous MacOS X, et a rendu l'installation sous Windows plus facile en le distribuant sous la forme d'une archive auto-installable pour ce système d'exploitation. Daniel de Rauglaudre a aussi complété la documentation du logiciel, en particulier en ajoutant un chapitre sur comment créer un CD-ROM contenant le logiciel et des bases de données.

GeneWeb est depuis cette année utilisé par la société GeneaNet, qui gère grâce à GeneWeb les bases de données généalogiques de ses utilisateurs (10 000 bases en novembre 2001). D'autres sites ou sociétés sont sur le point d'ouvrir des services similaires en utilisant GeneWeb.

GeneWeb est disponible depuis `http://cristal.inria.fr/~ddr/GeneWeb/`.

6.5.5 Concours ICFP

Participants : Sébastien Ailleret [projet Moscova], Sylvain Conchon [projet Moscova], Maxence Guesdon, Tom Hirschowitz, Fabrice Le Fessant [projet Sor], James Leifer [projet Moscova].

Un programme écrit en Objective Caml par Sébastien Ailleret, Sylvain Conchon, Maxence Guesdon, Fabrice Le Fessant, James Leifer et Tom Hirschowitz a obtenu le troisième prix au concours de programmation organisé à l'occasion du congrès ICFP 2001.

6.6 Compilation certifiée

Participants : Benjamin Grégoire, Xavier Leroy, Benjamin Werner [Logical].

Co-encadré par Xavier Leroy et Benjamin Werner (projet Logical), Benjamin Grégoire s'intéresse maintenant à la certification de compilateurs réalistes, avec pour objectifs d'une part d'introduire des mécanismes efficaces pour déterminer l'inter-convertibilité de termes dans le système Coq, et d'autre part d'ouvrir la voie à la certification du compilateur et de la machine abstraite d'OCaml.

Benjamin Grégoire a cette année réalisé pour le système Coq un compilateur et une machine abstraite fortement inspirés de la technologie d'Ocaml. Benjamin Grégoire a utilisé ces outils pour implanter un nouvel algorithme de mise en forme normale des termes Coq.

Enfin, il a réalisé une preuve de correction du compilateur et de la machine abstraite en Coq, pour un lambda-calcul avec déclarations locales.

6.7 Arithmétique

Participants : Robert Harley, Michel Mauny.

Robert Harley a quitté l'INRIA à la fin 2000, mais a néanmoins continué à travailler sur son sujet de thèse dont la soutenance aura lieu courant 2002. Robert Harley est encadré par Michel Mauny. Les travaux de Robert Harley ont porté plus particulièrement cette année sur l'algorithmique efficace de courbes elliptiques et des jacobiennes de courbes hyperelliptiques.

Les résultats les plus notables sont deux nouveaux algorithmes de calcul d'ordre de groupes algébriques en caractéristique deux, appelés « algorithme AGM » de genre 1 et 2.

6.7.1 Courbes elliptiques

Le premier algorithme, inventé avec Jean-François Mestre de l'Institut de Mathématiques de l'Université Paris 7, s'applique au cas des courbes elliptiques. En combinaison avec une stratégie *early-abort* efficace, développée avec Mireille Fouquet et Pierrick Gaudry du Laboratoire d'Informatique de l'École Polytechnique, il permet de calculer de nouvelles courbes sûres pour la cryptographie beaucoup plus rapidement qu'auparavant. Par exemple, cette méthode permet de trouver une bonne courbe de 200 bits en 5 secondes en moyenne sur une station Alpha de 750 MHz, alors qu'en 1999 le rapport ECDSA sur les signatures électroniques estimait que cela était une tâche qualifiée de « *complicated and cumbersome* », et nécessitant « *a few hours on a workstation* ».

Il est donc désormais possible d'augmenter la sécurité de la cryptographie elliptique, à taille de clé égale, en choisissant fréquemment des courbes sûres arbitraires, au lieu d'utiliser un petit nombre de courbes précalculées. Ceci permet de minimiser le risque associé au cassage éventuel de telle ou telle courbe par une nouvelle avancée mathématique, en distribuant le risque sur de nombreuses courbes. Dans certains cas il permet également de ne pas dévoiler publiquement la courbe utilisée, ce qui rend beaucoup plus difficile la tâche d'un adversaire qui voudrait s'attaquer au cryptosystème par un calcul de logarithme discret puisqu'il ne sait même pas à quel courbe s'attaquer.

6.7.2 Courbes hyperelliptiques

Le deuxième algorithme, developpé avec Mestre et Gaudry, est une méthode analogue pour le cas nettement plus compliqué des jacobiennes de courbes hyperelliptiques de genre deux et constitue le premier algorithme réellement efficace pour déterminer l'ordre de ces groupes.

Comme le premier algorithme, celui-ci permet de « relever » les courbes en question d'un corps fini vers un anneau 2-adique de caractéristique zéro en itérant le calcul explicite et

efficace d'une isogénie liée au « petit Frobenius » par la méthode de la moyenne arithmético-géométrique (*AGM* en anglais). Le relèvement calculé, dit « canonique », préserve certaines propriétés essentielles en particulier de l'endomorphisme de Frobenius, ce qui permet de déterminer ensuite l'ordre du groupe.

Ce domaine de recherche est devenu très actif depuis la présentation d'un premier algorithme efficace[Sat00] pour calculer le relèvement d'une courbe elliptique en petite caractéristique par le professeur Takakazu Satoh de l'Université de Saitama à la fin 1999. Actuellement le domaine évolue rapidement, surtout par l'extension des méthodes p-adiques à des courbes de plus en plus générales et par la réduction du temps de calcul asymptotique.

6.7.3 Présentations

En décembre 2000 et pendant 2001, les travaux décrits ci-dessus ont fait l'objet de plusieurs articles et présentations.

- En mai 2001, Robert Harley a participé à la conférence Eurocrypt 2001 (Innsbrück, Autriche), où il a présenté son travail [5]. Suite à plusieurs demandes de la part d'autres chercheurs, Robert Harley a aussi présenté très brièvement, au *Rump Session* d'Eurocrypt, quelques résultats obtenus en combinant *early-abort* avec le nouvel algorithme AGM dans le cas elliptique.
- En mai 2001, Jacques Stern a invité Jean-François Mestre et Robert Harley a venir présenter leur travail au Séminaire Complexité et Cryptographie de l'École Normale Supérieure, sous le titre « *Counting Points on Elliptic Curves with the Arithmetic-Geometric Mean* ». J-F. Mestre a décrit quelques concepts mathématiques autour de la moyenne arithmético-géométrique dans les nombres complexes, découverts au $19^{ième}$ siècle. R. Harley a décrit sommairement l'algorithme AGM et l'*early-abort* ainsi que les temps de calcul obtenus avec l'implémentation optimisée.
- En août 2001, au *Rump Session* de la conférence Crypto 2001 (Santa Barbara, Californie, États-Unis), Robert Harley a présenté le fonctionnement de l'algorithme AGM elliptique ainsi que les résultats obtenus.
- En octobre 2001, à la conférence *ECC 2001 – Elliptic Curve Cryptography* (Waterloo, Canada), Robert Harley a présenté en détail le fonctionnement de l'algorithme AGM dans le cas elliptique ainsi que les temps de calcul obtenus et les nouveaux records atteints, notamment le calcul d'une courbe de 16001 bits.
- En novembre 2001, à *MAGC 2001 – Midwest Arithmetical Geometry in Cryptography* (Urbana, Illinois, USA), Robert Harley a présenté un exposé invité décrivant en détail les aspects mathématiques des algorithmes AGM en genre 1 et 2 ainsi que les techniques d'implémentation.

Enfin, en ce moment, Robert Harley prépare un article détaillant les aspects théoriques et pratiques des algorithmes AGM en genre 1 et 2.

[Sat00] T. Satoh, « The Canonical Lift of an Ordinary Elliptic Curve over a Finite Field and its Point Counting », *Journal of the Ramanujan Mathematical Society 15*, décembre 2000, p. 247–270.

6.8 Linguistique computationnelle

Participants : Gérard Huet, Farhad Mehta.

Gérard Huet a poursuivi en 2001 ses contacts avec la communauté « Linguistique computationnelle ». Il a ainsi participé en juillet aux conférences internationales ACL'01 à Toulouse, et LACL 2001 au Croisic. Il a poursuivi ses travaux sur une plate-forme logicielle de manipulation du sanskrit et a écrit un article général sur l'architecture de la plateforme [21]. Gérard Huet a exposé ses travaux et présenté une démonstration au *Workshop on Computational Linguistics in South Asian Languages* de la *XXIth South Asian Languages Analysis Roundtable*, qui s'est tenue en octobre à l'Université de Konstanz [22].

Gérard Huet a mis au point une méthode générale d'analyse de relations rationnelles exprimant des règles d'euphonie, par compilation en transducteurs d'état fini. Cet algorithme permet notamment de faire l'analyse systématique du sandhi en sanskrit, et donc d'implanter un segmenteur. Cette méthode étant basée sur le lexique, elle permet de plus d'obtenir un étiqueteur morphologique. L'algorithme correspondant a donné lieu à la soumission d'un article de conférence [23] et d'un article de revue [24]. Ce travail a été présenté au séminaire du laboratoire LIAFA de l'Université Paris 7 en octobre.

7 Contrats industriels (nationaux, européens et internationaux)

7.1 France Telecom R&D : Flots synchrones en ML

Le travail de Pascal Cuoq sur l'adjonction de synchronisme dans les langages fonctionnels est partiellement financé par France Telecom R&D.

7.2 Consortium Caml

Le Consortium Caml a pour vocation de réunir les industriels et académiques désireux d'aider au développement et de pérenniser le langage Caml et les outils associés. En novembre 2001, le Consortium Caml comprenait 5 membres industriels, et une réunion préliminaire s'est tenue à Rocquencourt au début juillet 2001. Le site web du consortium est `http://caml.inria.fr/consortium/` et la première assemblée des membres du Consortium est prévue au début 2002.

8 Actions régionales, nationales et internationales

8.1 Actions nationales

8.1.1 ACI GRID

Les projets Cristal et Moscova participent à une Action Concertée Incitative GRID, intitulée « PL4 - CARAML », et avec pour partenaires le LIFO (Univ. d'Orléans), PPS (Univ. Paris 7), le LACL (Univ. Paris 12). L'action s'étale sur 3 ans, et est dotée d'un financement de 765KF.

L'ACI est coordonnée par Gaétan Hains, du LIFO.

8.1.2 GDR ALP

Michel Mauny est membre du conseil scientifique du GDR *Algorithmique, Langage et Programmation* (ALP).

8.2 Actions financées par la Commission Européenne

8.2.1 Groupe de travail Esprit *Applied Semantics*

Le projet Cristal participe au groupe de travail Esprit 26142 *Applied Semantics*, dont le but est de réunir théoriciens et spécialistes en langages de programmation afin de concentrer les travaux théoriques en sémantique de langages sur des aspects pratiques importants. Le groupe a été créé courant 1997, et son aspect interdisciplinaire est l'une de ses caractéristiques principales. Il a récemment été prolongé jusqu'en mars 2002.

Le coordinateur du groupe est l'université de Chalmers (Suède), et le groupe réunit bien sûr des équipes académiques européennes (Danemark, Royaume Uni, Suède, France, Italie), mais aussi quelques industriels.

8.3 Relations bilatérales internationales

8.3.1 Inde

Une équipe associée franco-indienne en Traitement Informatique de la Langue a été acceptée en septembre 2001 pour 3 ans. Le correspondant français est Gérard Huet, le correspondant indien est le Pr Narayana Murthy de l'Université d'Hyderabad. Voir `http://pauillac.inria.fr/~huet/FIRNCL/`.

9 Diffusion de résultats

9.1 Animation de la communauté scientifique

9.1.1 Animation interne à l'INRIA

Xavier Leroy est membre titulaire élu de la Commission d'Évaluation de l'INRIA. À ce titre, il a fait partie du jury de recrutement CR1, du jury de recrutement/promotion DR2, des sections locales d'audition CR2 des unités de Rennes et de Sophia, et des jurys de promotion CR1, DR1 et DR0, tout ceci ayant entraîné l'écriture de 27 rapports sur des candidatures. Il a également assisté au séminaire d'évaluation du programme 3A (octobre 2001), et a été délégué de la Commission d'Évaluation pour la création des projets Vertecs, Spaces et Miró.

Michel Mauny est membre élu (suppléant) de la Commission d'Évaluation. Il a présidé la Section Locale d'Audition du concours CR2 2001 de l'UR de Rocquencourt (64 dossiers), et a participé au jury de recrutement DR2 de l'INRIA. Il est membre de l'équipe de direction de l'UR, et a été membre du groupe de travail sur le logiciel libre organisé par la DirDRI. Il a donné un exposé sur la diffusion d'OCaml à l'occasion de deux formations internes à l'INRIA (à Rennes et à Nancy) sur le thème de la valorisation.

Didier Rémy est membre du Comité des bourses de l'UR de Rocquencourt.

Avec Michel Loyer pour la dimension administrative, Pierre Weis est co-responsable des moyens informatiques de Rocquencourt depuis mai 1999. Au sein de ce tandem, Pierre Weis s'occupe des bonnes relations avec les chercheurs : il recueille les besoins des projets et facilite l'établissement et le suivi des bons de commande. Cette année, Pierre Weis a lancé une opération de portage du logiciel de réseau sans fil Hyperlan (produit du projet Hipercom). François Péron (service Myriad) et Cédric Adjih (projet Hipercom) sont les principaux artisans de cet effort. Cédric Adjih a déjà porté le logiciel sur Windows2000 et s'apprête à le porter sur les assistants personnels StrongArm/WinCE et Linux (Ipaq, HP Jornada).

Le « délégué communication relations chercheurs » de l'UR de Rocquencourt est formé d'un triumvirat comprenant Daniel Augot (projet Codes), Éliane Bécache (projet Ondes) et Pierre Weis. Dans ce cadre, Pierre Weis a participé à de nombreuses réunions concernant la réfection de la plaquette de Rocquencourt ; avec Daniel Augot, Pierre Weis s'est chargé en particulier de recueillir et de rédiger avec les chefs de projets les textes concernant les programmes 1 et 2.

Pierre Weis est membre élu du Conseil Scientifique de l'INRIA et membre du Comité d'UR de Rocquencourt.

9.1.2 Animation de la communauté scientifique hors INRIA

Gérard Huet fait partie du *Board de l'International Institute for Software Engineering* de l'Université des Nations Unies à Macao.

Gérard Huet est responsable pour l'INRIA des relations scientifiques avec l'Inde. A ce titre il s'est rendu en Inde en mars et en novembre, et il organise un programme d'échanges d'étudiants avec les IITs (une quarantaine de stages d'été de 3 mois en 2001).

Xavier Leroy est membre du groupe de travail IFIP 2.8 sur la programmation fonctionnelle.

Michel Mauny travaille régulièrement avec la Direction des Relations Internationales de l'INRIA. Il est membre du Comité Scientifique de la coopération franco-marocaine dans le domaine des STIC (programme géré par l'INRIA du côté français). Il a organisé un workshop sur le thème du Génie Logiciel en mai à Rabat où il a donné un exposé sur le typage statique. L'objectif de ce workshop était de susciter des propositions de collaborations franco-marocaines. Il s'est rendu une seconde fois au Maroc en octobre 2001 pour une réunion de ce comité durant laquelle ont été examinées ces propositions. Il a participé à la rencontre INRIA-HKU (Hong Kong University) où il a donné un exposé de présentation d'Objective Caml et participé à une table ronde sur le logiciel libre. Cet événement était associé à la signature d'un *memorandum of understanding* entre l'INRIA et HKU.

Didier Rémy est membre du comité de direction du workshop *Fondations of Object Oriented Languages (FOOL).*

9.1.3 Animation de la communauté des usagers de Caml

Participants : Maxence Guesdon, Pierre Weis.

Le Caml Hump Mxence Guesdon a remis à jour et maintient les pages web du *Caml Hump*, regroupant les différents outils, applications, et bibliothèques relatifs à Caml et OCaml. La gestion du *Hump* utilise maintenant quelques-uns des outils qu'a développés Maxence Guesdon.

Formation Pierre Weis a réalisé un recueil d'exemples pour la programmation en Caml (programmes en Caml Light et Objective Caml) qui est disponible sur le site Web du langage (plusieurs dizaines de programmes).

Pierre Weis continue à suivre et à encourager les efforts des professeurs des classes préparatoires. Il s'est rendu à Marseille au colloque de l'association des professeurs de mathématiques des classes préparatoires, du 2 au 5 mai 2001, où il a fait un cours d'introduction à l'interface graphique Caml/Tk.

9.1.4 Comités de lecture et programmes

Xavier Leroy est éditeur associé de la revue *Journal of Functional Programming*. Il a présidé le comité de programme du congrès International Conference on Functional Programming 2001. Il a également fait partie du comité de programme du symposium Principles of Programming Languages 2001.

Michel Mauny est membre du comité de rédaction de Techniques et Sciences Informatiques.

François Pottier a été membre des comités de programmes des conférences *Theoretical Aspects of Computer Software* (TACS'01) et *Journées Francophones des Langages Applicatifs* (JFLA'02).

Didier Rémy a été membre du comité de programme de l'*European Symposium On Programming (ESOP)* qui aura lieu en avril 2002 à Grenoble.

9.1.5 Jurys de thèses

Gérard Huet a été rapporteur de la thèse de Sylvain Pogodalla, soutenue à l'Institut Polytechnique de Lorraine en septembre. Il a présidé le jury de thèse de Samuel Lacas à l'Université Paris 7 en juin.

Xavier Leroy a été rapporteur de la thèse de Laurent Réveillère (Université de Rennes 1, décembre 2001).

Didier Rémy a été rapporteur de la thèse de Fabien Dagnat, soutenue à l'Institut National Polytechnique de Toulouse en mai 2001.

9.1.6 Autres activités d'intérêt général

Diffusion des connaissances **Participants** : Roberto Di Cosmo, Pierre Weis.

Roberto Di Cosmo contribue depuis plusieurs années à la diffusion des logiciels libres, par une activitée de vulgarisation auprès du grand public. Il est aussi le fondateur du projet DemoLinux (auquel participent Vincent Balat (PPS) et Jean-Vincent Loddo (PPS)), accessible sur le site web `http://www.demolinux.org/`. Ce projet a pour but la réalisation de CD-ROMs capables de faire démarrer un ordinateur sans utilisation du disque dur, tout en offrant un système Linux (Unix) complètement opérationnel, ce qui permet de diffuser aisément des logiciels scientifiques à la configuration complexe, et cela à un très large public.

La version 3.0 est disponible depuis le 16 octobre 2001, et incorpore déjà deux des langages développés à l'INRIA : Ocaml (version 3.02) et Elan.

Pierre Weis a réalisé une cinquième version du CD-ROM de diffusion des logiciels de l'INRIA, « Logiciels libres disponibles à l'INRIA ». Ce CD-ROM comprend tous les logiciels du projet et ceux de tous les projets de l'INRIA qui ont répondu à cette initiative (plus de 30 logiciels). Le CD-ROM a été diffusé gratuitement à 1500 exemplaires cette année. La sixième version de ce CD-ROM est en cours de réalisation et devrait sortir au début du mois de janvier 2002.

9.2 Enseignement

9.2.1 Encadrement et jurys

Roberto Di Cosmo encadre la thèse de Vincent Balat et Jean-Vincent Loddo, tous deux au laboratoire PPS.

Xavier Leroy encadre la thèse de Tom Hischowitz et co-encadre celle de Benjamin Grégoire (avec Benjamin Werner, projet Logical).

Michel Mauny encadre la thèse de Robert Harley et co-encadre celle de Pascal Cuoq (avec Marc Pouzet, LIP6).

Didier Rémy encadre les thèses de Didier Le Botlan et Daniel Bonniot depuis septembre 2000.

Pierre Weis encadre la thèse de Jun Furuse.

9.2.2 Enseignements de troisième cycle universitaire

Xavier Leroy enseigne le cours « Typage et programmation » du tronc commun du DEA « Programmation : sémantique, preuves et langages » (20h). Il est responsable de la filière « Langages et applications » de ce DEA.

Roberto Di Cosmo enseigne avec Delia Kesner le cours d'option « Logique Linéaire et Calcul des Motifs » du DEA « Programmation : sémantique, preuves et langages » (20h).

François Pottier a assuré trois séances (6h) du cours *Typage et sous-typage* de Giuseppe Castagna, dans le cadre du DEA « Programmation : sémantique, preuves et langages ».

9.2.3 Enseignement en écoles d'ingénieurs

Didier Rémy est professeur chargé de cours à temps partiel à l'École Polytechnique. À ce titre il a enseigné un cours de compilation et un cours sur la modularité dans le cursus de deuxième année.

Michel Mauny a donné un cours sur OCaml (17 heures) à l'Institut Supérieur d'Informatique et d'Automatique (ISIA) à Sophia Antipolis à des élèves en dernière année.

Benjamin Gregoire a donné des TP de Java à l'École Polytechnique, ainsi que des TP de compilation à Orsay.

9.2.4 Enseignements de premier et second cycles universitaires

Dans le cadre de leur monitorat à l'Université Paris 7 :

– Didier Le Botlan a assuré un total de 60 heures à l'université Paris 7, en qualité de responsable de TD de géométrie et de TP de C++ ;

– Tom Hirschowitz a donné des travaux pratiques et dirigés de structures de données en C++, en cours de DEUG première année à l'Université Paris 7 (43 heures).

Vincent Simonet a encadré les travaux pratiques d'option informatique en classes préparatoires MPSI au Lycée Janson-de-Sailly (programmation Caml).

9.2.5 Autres enseignements

Avec l'aide de Bruno Verlyck pour les travaux dirigés, Pierre Weis a fait un cours d'Objective Caml très complet (3 fois deux jours) aux chercheurs numériciens et aux ingénieurs volontaires de l'INRIA Rocquencourt.

9.3 Participation à des colloques, séminaires, invitations

Gérard Huet a été invité à présenter une Distinguished Lecture pour le 90ème anniversaire de la Faculty of Engineering of the University of Hong Kong, le 31 mai.

Xavier Leroy a présenté un exposé invité sur la vérification de bytecode Java et ses liens avec les méthodes formelles [8] au congrès Computer Aided Verification (Paris, juillet 2001).

Xavier Leroy a présenté un exposé invité sur le rôle du typage statique et des analyses automatiques de programmes dans la sûreté du logiciel [25] aux journées de l'Association Française d'Informatique Théorique (Paris, avril 2001).

Daniel Bonniot, Xavier Leroy, Didier Le Botlan et Didier Rémy ont participé à la conférence *Principle of Programming Languages (POPL)* qui s'est tenue à Londres en janvier 2001. Didier Rémy, Daniel Bonniot et Didier Le Botlan ont aussi participé au workshop satellite *Fondations of Object Oriented Languages (FOOL)* qui a eu lieu à cette occasion.

Tom Hirschowitz, Didier Le Botlan, Xavier Leroy, François Pottier et Didier Rémy ont assisté à la conférence ICFP'01 à Florence (septembre 2001).

Pascal Cuoq et François Pottier ont assisté à la conférence ESOP'01 à Gênes en avril 2001. Pascal Cuoq y a présenté son travail sur l'analyse de causalité [4]. François Pottier y a présenté un travail commun avec Sylvain Conchon concernant l'inférence de types à base de contraintes pour le *join*-calcul [3]. Un autre article [11], cosigné par François Pottier, Christian Skalka et Scott Smith (John Hopkins University, Baltimore) a également été présenté à cette conférence, et a reçu une distinction (*ETAPS'01 best paper award*) de la part de l'*European Association for Programming Languages and Systems* (EAPLS).

Xavier Leroy et Michel Mauny ont participé au Symposium on Static Analysis (SAS, Paris, juillet 2001).

Gérard Huet a participé en juillet aux conférences internationales ACL'01 à Toulouse, et LACL 2001 au Croisic. Il s'est rendu à Konstanz en octobre pour donner un exposé et présenter une démonstration au *Workshop on Computational Linguistics in South Asian Languages* de la *XXIth South Asian Languages Analysis Roundtable*. Gérard Huet a d'autre part présenté l'utilisation d'Ocaml et de Camlp4 dans les applications de rétro-ingéniérie à la conférence IEEE WCRE'2001 à Stuttgart en octobre [7].

François Pottier et Didier Rémy ont participé à la réunion du *Working Group* 2.8 de l'IFIP qui s'est tenu à Åre en Suède du 22 au 27 avril 2001. Didier Rémy y a présenté son travail sur les objets dans le calcul *Join*.

Xavier Leroy a présenté son article sur la vérification de bytecode Java embarquée sur cartes à puces au congrès E-Smart (Cannes, septembre 2001).

Michel Mauny et Didier Rémy ont participé au workshop *Dependent Types in Programming* qui s'est tenu à Dagstuhl du 19 au 24 août. Didier Rémy y a présenté ses travaux avec Didier Le Botlan sur l'inférence de types dans un système mixte ML^F.

Pascal Cuoq, Jun Furuse, Xavier Leroy, Michel Mauny et Pierre Weis ont participé aux Journées Francophones des Langages Applicatifs (Pontarlier, janvier 2001).

Tom Hirschowitz a effectué une visite de quelques jours à l'Université de Gènes en Italie en avril pour y travailler avec Davide Ancona et Elena Zucca. En octobre, il a passé deux semaines à l'Université de Heriot-Watt en Écosse, en réponse à l'invitation de Joe Wells pour travailler avec ce dernier sur le sujet des mixins.

Gérard Huet, en temps que membre du *Board de l'International Institute for Software Engineering* s'est rendu à Macao en mai.

Gérard Huet a donné un exposé au séminaire du LIAFA à l'Université Paris 7 en octobre.

Michel Mauny s'est rendu au workshop franco-marocain qu'il organisait sur les thèmes du génie logiciel (Rabat, mai 2001) et il y a donné un exposé. Il s'est de nouveau rendu au Maroc en novembre pour participer à une réunion du comité scientifique de la coopération franco-marocaine sur les STIC à Rabat, ainsi qu'au workshop organisé à Fès par une action de recherche franco-marocaine.

François Pottier a rendu une visite de quelques jours au Laboratoire des Méthodes de Programmation de l'École Polytechnique Fédérale de Lausanne, dirigé par Martin Odersky (janvier 2001).

François Pottier et Vincent Simonet se sont rendus à l'INRIA Sophia (septembre 2001). Vincent Simonet y a exposé leurs travaux concernant l'analyse de flots d'information, en présence de Gérard Boudol, Ilaria Castellani, Kohei Honda et Nobuko Yoshida.

François Pottier a présenté un exposé sur l'inférence de types à base de contraintes à l'IRIT (Université Paul Sabatier, Toulouse, novembre 2001).

Vincent Simonet s'est rendu à l'IRISA (octobre 2001) à l'invitation de Thomas Jensen. Il y a exposé ses travaux dans le cadre du séminaire du projet Lande.

Vincent Simonet a présenté ses travaux au séminaire de l'équipe Démons du L.R.I. d'Orsay (novembre 2001).

9.4 Relations industrielles

Xavier Leroy est consultant (1 jour par semaine) auprès de la société Trusted Logic dans le cadre d'une convention de concours scientifique.

Pierre Weis est consultant (1 jour par semaine) auprès de la société LexiFi Technologies.

10 Bibliographie

Articles et chapitres de livre

[1] M. Fouquet, P. Gaudry, R. Harley, « An extension of Satoh's algorithm and its implementation », *Journal of the Ramanujan Mathematical Society 15*, 2000, p. 281–318.

[2] X. Leroy, « Bytecode verification on Java smart cards », *Software : Practice and Experience*, 2001, Accepté pour publication, à paraître.

Communications à des congrès, colloques, etc.

[3] S. Conchon, F. Pottier, « JOIN(X) : Constraint-Based Type Inference for the Join-Calculus », *in : Proceedings of the 10th European Symposium on Programming (ESOP'01)*, D. Sands (éditeur), *Lecture Notes in Computer Science, 2028*, Springer Verlag, p. 221–236, avril 2001.

[4] P. Cuoq, M. Pouzet, « Modular Causality in a Synchronous Stream Language », *in : Proceedings of the 10th European Symposium on Programming (ESOP'01)*, D. Sands (éditeur), *Lecture Notes in Computer Science, 2028*, Springer Verlag, p. 30–45, avril 2001.

[5] M. Fouquet, P. Gaudry, R. Harley, « Finding Secure Curves with the Satoh-FGH Algorithm and an Early-Abort Strategy », *in : Advances in Cryptology – Eurocrypt 2001*, B. Pfitzmann (éditeur), *2045*, Lecture Notes in Computer Science, p. 14–29, mai 2001.

[6] J. Furuse, « Generic Polymorphism in ML », *in : Actes des Journées francophones des langages applicatifs*, INRIA, janvier 2001.

[7] G. Huet, « From an informal textual lexicon to a well-structured lexical database : An experiment in data reverse engineering », *in : Working Conference on Reverse Engineering (WCRE'2001)*, IEEE, 2001.

[8] X. Leroy, « Java bytecode verification : an overview », *in : Computer Aided Verification, CAV 2001*, G. Berry, H. Comon, A. Finkel (éditeurs), *Lecture Notes in Computer Science, 2102*, Springer-Verlag, p. 265–285, 2001. Exposé invité, `http://pauillac.inria.fr/~xleroy/publi/survey-bytecode-verification.ps.gz`.

[9] X. Leroy, « On-card Bytecode Verification for Java Card », *in : Smart card programming and security, proceedings E-Smart 2001*, I. Attali, T. Jensen (éditeurs), *Lecture Notes in Computer Science, 2140*, Springer-Verlag, p. 150–164, 2001, `http://pauillac.inria.fr/~xleroy/publi/oncard-verifier.ps.gz`.

[10] F. Pottier, V. Simonet, « Information Flow Inference for ML », *in : accepté pour présentation au 29th ACM Symposium on Principles of Programming Languages, Portland, Oregon*, janvier 2002.

[11] F. Pottier, C. Skalka, S. Smith, « A Systematic Approach to Static Access Control », *in : Proceedings of the 10th European Symposium on Programming (ESOP'01)*, D. Sands (éditeur), *Lecture Notes in Computer Science, 2028*, Springer Verlag, p. 30–45, avril 2001.

[12] J. Vouillon, « An object calculus with views », *in : Proceedings 28th ACM symposium Principles of Programming Languages*, ACM Press, janvier 2001, `http://cristal.inria.fr/~vouillon/publi/views.ps.gz`.

Rapports de recherche et publications internes

[13] D. de Rauglaudre, *Camlp4 – Reference Manual version 3.03*, INRIA, novembre 2001, documentation distribuée avec le système Objective Caml, `http://caml.inria.fr/camlp4/manual/`.

[14] D. de Rauglaudre, *Camlp4 – Tutorial version 3.03*, INRIA, novembre 2001, documentation distribuée avec le système Objective Caml, `http://caml.inria.fr/camlp4/tutorial/`.

[15] M. Guesdon, *OCamldoc documentation and user's manual – release 3.03*, INRIA, décembre 2001, documentation distribuée avec OCamldoc.

[16] X. Leroy, D. Rémy, J. Garrigue, J. Vouillon, D. Doligez, *The Objective Caml system, documentation and user's manual – release 3.02*, INRIA, juillet 2001, documentation distribuée avec le système Objective Caml, `http://caml.inria.fr/ocaml/htmlman/`.

[17] F. Pottier, « A semi-syntactic soundness proof for HM(X) », *Rapport de Recherche n°4150*, INRIA, mar 2001, `http://www.inria.fr/rrrt/rr-4150.html`.

Divers

[18] D. Bonniot, « Type-checking multi-methods in ML (A modular approach) », soumis à publication, octobre 2001.

[19] T. Hirschowitz, X. Leroy, « Mixin modules in a call-by-value setting », Soumis à publication, 2001.

[20] T. Hirschowitz, X. Leroy, « Mixin modules in a call-by-value setting », Version longue de [19], en préparation, 2001.

[21] G. Huet, « Computational Linguistics for Sanskrit : a Software Engineering Approach », soumis à publication, 2001.

[22] G. Huet, « Computational Tools for Sanskrit », 2001, Workshop on Computational Linguistics in South Asian Languages, XXIth South Asian Languages Analysis Roundtable.

[23] G. Huet, « Reversible Junction Transducers », soumis à publication, 2001.

[24] G. Huet, « Tagging a Stream of Phonems by Lexicon-directed Euphony Analysis – Application to a Sanskrit Lemmatizer », soumis à publication, 2001.

[25] X. Leroy, « Où va la programmation ? ou : Des programmes pour rendre les programmes plus fiables », avril 2001, exposé invité aux journées de l'Association Française d'Informatique Théorique.