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 (commun avec COQ)

Nelly Maloisel

Personnel INRIA

Xavier Leroy [CR]

Daniel de Rauglaudre [IR]

François Pottier [CR, depuis oct. 1999]

Didier Rémy [DR]

Collaborateurs extérieurs

Manuel Serrano [Maître de Conférences, Univ. de Nice]

Chercheur invité

Jacques Garrigue [RIMS, Univ. de Kyoto, Japon, d'octobre à fin décembre 1999]

Ingénieur expert

Bruno Pagano [depuis sept. 1999]

Doctorants

Pascal Cuoq [Boursier INRIA, Univ. Paris 6]

Jun Furuse [Boursier du gouvernement Japonais, Univ. Paris 7]

Robert Harley [Allocataire MENRT, Univ. Paris 7]

François Pessaux [Boursier INRIA, Univ. Paris 6]

Jérôme Vouillon [AMN, Univ. Paris 7]

Stagiaires

Benjamin Grégoire [Stagiaire de DEA, Univ. Paris 7, commun avec COQ]

Sylvain Peyronnet [Stagiaire de Maîtrise, Univ. Paris 7]

2 Présentation et objectifs généraux

Le projet Cristal s'intéresse aux formalismes de typage statique 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 montrant la nécessité d'environnements de programmation typée alliant fiabilité, sécurité et efficacité.

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

3 Fondements scientifiques

3.1 Systèmes de types

Glossaire :

Typage Discipline de programmation visant à vérifier et garantir l'utilisation cohérente des données manipulées dans les programmes.

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

Synthèse de types On dit que les types des programmes sont synthétisés (ou encore inférés) lorsque l'on n'impose pas au programmeur de spécifier les types de toutes les expressions du programme.

Polymorphisme Un type est dit polymorphe lorsque certaines de ses composantes sont variables, et représentent toute une famille de types (obtenus par instanciation de ces variables).

Résumé : *Le typage est une discipline importante de la programmation: il permet de spécifier la cohérence des données manipulées par les programmes, et de définir certains mécanismes de structuration du code (fonctions ou procédures, objets, modules, surcharge) propres aux langages de haut niveau à même d'être utilisés pour la construction d'applications complexes.*

L'équipe Cristal étudie les systèmes de types et les méthodes formelles y afférant, avec pour objectif de permettre la conception de langages et autres 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 autorisant la synthèse automatique des types à la compilation, ainsi que d'une forme de polymorphisme permettant la programmation d'algorithmes génériques (i.e. travaillant sur 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 à l'exécution.

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 ont été réalisées avec pour principal objectif d'accroître l'expressivité du langage tout en en préservant les propriétés essentielles,
- les méthodes développées à cette occasion ont souvent été réutilisées et enrichies pour être appliquées à d'autres études sortant éventuellement du cadre strict du langage ML.

Les résultats de nos recherches dans ce domaine, même si celles-ci ont une vocation plus générale, trouvent généralement des applications dans le langage Caml: une version de ML conçue et développée dans notre équipe.

Dans le domaine des systèmes de types, les sujets actuellement étudiés vont des fondements de la programmation par objets à la sécurité des programmes, en passant par différentes formes de polymorphisme, allant même jusqu'à l'utilisation des formalismes de typages dans des analyses statiques dépassant le cadre des systèmes de types. Ces différents sujets sont détaillés dans les sections suivantes.

3.1.2 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 utilisées avec pour but d'établir les fondements de la programmation par objets en termes de théorie des types.

L'une des approches que nous suivons au projet Cristal repose sur les travaux de Didier Rémy, qui a proposé dans sa thèse un système d'enregistrements extensibles supportant 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'adjonction d'objets dans le langage Caml, devenu ainsi Objective Caml (OCaml). Cette couche à objets, développée en collaboration avec Jérôme Vouillon, s'appuyant sur des classes et compatible avec la synthèse

[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.

de types polymorphes, est l'une des plus attrayantes parmi celles proposées pour les langages fortement typés, en dépit de la contrainte imposée par la synthèse des types. Par exemple, le système OCaml permet de typer des programmes réputés difficiles et rejetés par le langage Java, pourtant explicitement typé. C'est le cas notamment des programmes nécessitant, dans le jargon Java, des «types virtuels»; ce travail pourrait, en ce sens, servir de base à la formalisation des types virtuels de Java.

Une autre étude poursuivie est le sous-typage implicite, qui fut le sujet de thèse de François Pottier, et qui permettrait entre autre, de rendre OCaml encore plus performant et plus convivial, et qui fournit des résultats suffisamment généraux pour aborder d'autres problèmes que celui du typage (par exemple, l'optimisation ou les problèmes de sécurité). Une dernière approche explorée repose sur la surcharge d'identificateurs et le comportement de fonctions guidé par le type de leurs arguments: c'est le polymorphisme extensionnel, sujet de thèse de Jun Furuse.

3.1.3 Typage et analyses statiques

Il est aussi possible d'aborder l'analyse et la compilation optimisante de programmes en utilisant des techniques qui relèvent du typage. C'est le cas de l'analyse d'exceptions, qui devrait fournir à l'usager, dès la compilation, des informations précieuses sur les exceptions qui peuvent être levées durant l'exécution des programmes. François Pessaux effectue une thèse sur ce sujet. De même, les informations fournies par le typage statique sont une aide précieuse pour produire du code efficace lors de la compilation. Xavier Leroy a ainsi développé des techniques de représentations efficaces de structures de données qui reposent de manière essentielle sur le typage statique.

3.1.4 Typage et sécurité

L'effervescence autour du code mobile et la programmation des cartes à puce suscite un regain d'intérêt pour la sûreté et la sécurité des programmes. Les méthodes de typage fournissent un angle nouveau pour aborder ces problèmes dans ce contexte. Le travail de Xavier Leroy et François Rouaix a constitué une première approche et a ouvert une voie vers des techniques formelles pour développer et valider des environnements d'exécution sûrs pour des *applets* dans un langage fortement typé. Benjamin Grégoire (membre des projets Cristal et Coq) effectue une thèse sur ce sujet, co-encadré par Xavier Leroy et Benjamin Werner (projet Coq).

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 les fonctions, presque au sens mathématique du terme: un programme est une expression à évaluer qui fournit un résultat dépendant seulement des valeurs des paramètres d'entrée.

Plus classique, le style **impératif** privilégie l'exécution d'instructions qui modifie l'état de la machine (sa mémoire ou bien ses entrées-sorties).

Le style **à objets** favorise le regroupement de données, de leur état (sorte de mémoire privée) et de leurs comportements (méthodes) en des entités appelées objets. 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 est appelé héritage. De la sorte, seules les fonctionnalités nouvelles doivent être programmées. 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 Un compilateur de langage de programmation traduit les programmes écrits dans ce langage en des instructions de plus bas niveau, interprétables par la machine. Les instructions peuvent être du **bytecode** fonctionnant sur une grande variété de machines, ou bien du **code natif** spécifique à l'architecture de la machine sous-jacente. Le bytecode peut être le même pour une grande variété d'architectures de machines et est interprété par un programme dédié appelé **machine virtuelle**. Le code natif est directement interprété par la machine, et dépend donc de son architecture.

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

Résumé : *Le langage de programmation Caml est l'un des langages de la famille ML. Tout comme les autres versions de ML, Caml a été à la fois la source d'inspiration et l'objet de nombreuses recherches, et s'est souvent enrichi des solutions apportées, 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 est nommée 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, l'utilise comme champ d'expérimentation et de validation, ainsi que comme moyen de transfert et de diffusion de nos résultats.

Le langage Caml est l'héritier des premières versions de ML qui était le méta-langage de l'assistant de preuves LCF conçu par Robin Milner. ML était le langage dans lequel étaient programmées les tactiques de recherche de preuves de 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. En particulier, deux branches distinctes

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

de ML sont apparues au milieu des années 80: l'une, nommée Standard ML, est le résultat du travail de conception et de définition de ML par un groupe à la tête duquel se trouvait Robin Milner. L'une des caractéristiques les plus novatrices de Standard ML était son système de modules.

Le langage Caml constitue l'autre branche de la famille des langages ML. Conçu et développé au 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 des évolutions importantes.

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 dû être l'objet de recherches intensives afin d'être dotés de sémantiques statiques (typage) et dynamiques (exécution) clairement définies et prouvées correctes. Les objets ont été étudiés par Didier Rémy (cf. section 3.1.2). Les modules ont été étudiés par Xavier Leroy [Ler96], qui a étendu et simplifié le système de modules de Standard ML afin de 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 incorporant tous ces traits est nommée Objective Caml.

Du point de vue de l'implémentation, le langage Caml a donné lieu à des recherches sur les modèles d'exécution des langages fonctionnels. Le modèle d'exécution de Caml, initialement basé sur la Machine Abstraite Catégorique (CAM) et s'appuyant sur la machine LLM3 de Le-Lisp (de l'INRIA), a été changé en machine virtuelle bytecodée avec l'implémentation Caml-Light [LW93]. Cette implémentation, réalisée par Xavier Leroy, et s'appuyant sur un gestionnaire de mémoire dû à Damien Doligez [DL93] a été conçue de sorte à être portable et économe en ressources mémoire. Lors du passage de Caml Light à Objective Caml, Xavier Leroy a intégré un compilateur produisant du code natif performant qui, allié au générateur de bytecode, permet dorénavant de disposer du meilleur des deux mondes: portabilité et cycle de développement rapide grâce au bytecode, et code natif très efficace avec le compilateur optimisant (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 de permettre la détection rapide d'erreurs de programmation. Les recherches dans ce domaine ont donc comme (vaste) champ d'application la programmation en général. Lorsqu'on allie à cette détection d'erreurs des facilités de structuration du code telles les fonctions, les modules ou les objets, c'est plus précisément l'abstraction et la réutilisabilité que l'on cherche à améliorer, permettant ainsi l'évolution contrôlée de systèmes complexes.

[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.

[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.

De fait, plusieurs expériences ont montré que les développements réalisés dans un langage de programmation tel 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 peuvent s'appliquer 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 s'exécutant 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 Enseignement de la programmation

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

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 accessibles à l'URL `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 oeuvre par Daniel de Rauglaudre

d'un préprocesseur du langage Caml nommé Camlp4. Camlp4 permet à l'utilisateur de substituer son propre analyseur syntaxique à celui d'Objective Caml, permettant ainsi d'étendre le langage ou bien de le restreindre, et de le spécialiser à telle ou telle application.

Camlp4 est distribué gratuitement sous forme source et binaire, et sa licence est inspirée de la licence BSD.

6 Résultats nouveaux

6.1 Systèmes de types

6.1.1 Typage des objets

Participants : Didier Rémy, Jérôme Vouillon.

En collaboration avec Jérôme Vouillon, Didier Rémy s'est intéressé à la notion de *type virtuel* introduite dans des langages faiblement typés puis reprise par une partie de la communauté Java dans un contexte typé. Cette construction a parfois été considérée comme essentielle pour résoudre des problèmes réputés difficiles. Pourtant nous avons montré qu'elle peut être vue comme un simple cas particulier des classes paramétriques dans des langages ayant un système de types suffisamment puissant. C'est en particulier le cas dans le langage OCaml [11].

Didier Rémy a aussi étudié deux nouvelles formes de types enregistrements. Ces travaux sont récents et encore prospectifs.

Les enregistrements multi-dimensionels sont des tableaux multi-dimensionels à champs nommés hétérogènes. À la différence des enregistrements d'enregistrements, qui fixent un ordre de projection, les enregistrements multi-dimensionels permettent la commutation entre les différentes projections. Par exemple, dans un langage à objets, la table des méthodes peut être vue comme un tableau à deux dimensions donnant pour chaque classe et chaque message la fonction à exécuter à l'appel. L'avantage de cette approche est de considérer simultanément cette table comme un ensemble de classes définissant chacune un ensemble de méthodes ou un ensemble de méthodes définie chacune sur un ensemble de classe: on peut alors ajouter une nouvelle méthode (à un ensemble de classes) ou une nouvelle classe (avec une ensemble de méthodes). Cette solution permettrait de typer les fonctions génériques de Scheme et de corriger la dissymétrie créer par l'approche «objets comme enregistrements de méthodes».

Les enregistrements peuvent aussi être étendus en donnant de la structure aux étiquettes plutôt que de les traiter comme des atomes, la structure d'arbres étant un cas particulier intéressant. Une application ambitieuse serait de représenter la hiérarchie des classes dans les étiquettes et de pouvoir définir des méthodes simultanément pour une classe et ses sous-classe.

Jérôme Vouillon a étudié différents calculs d'objets [12] permettant d'avoir un système de classes plus souple que celui d'Objective Caml, donnant notamment la possibilité de masquer *a posteriori* des méthodes d'une classe. Cela n'est pas possible dans Objective Caml et nuit à la modularité du système de classes. En effet, le programmeur n'a aucun contrôle sur la liste des méthodes héritées par classe, et donc, par exemple, si une méthode est rajoutée dans une

classe parente au cours du développement d'un programme, il ne lui est pas possible d'éviter simplement une éventuelle collision avec une autre méthode de même nom déjà existante.

6.1.2 Objets et concurrence

Participants : Cédric Fournet [projet Para/ Microsoft Cambridge], Cosimo Laneve [Univ. de Bologne], Luc Maranget [projet Para], Dicier Rémy.

En collaboration avec Cédric Fournet (projet Para), Cosimo Laneve (Université de Bologne) et Luc Maranget (projet Para), Didier Rémy a poursuivi ses travaux sur l'adjonction au calcul Join d'objets et de classes primitifs. Dans un premier temps il s'agit d'un simple transfert de technologie issue du mécanisme de typage du calcul Join et du typage des objets et des classes dans le langage OCaml. On obtient ainsi une extension simple du calcul Join avec des objets et des classes. Mais la difficulté de ces travaux ainsi que leur réel intérêt résultent de l'interaction entre concurrence et héritage. Nous sommes amenés à ajouter de nouvelles constructions pour permettre l'héritage de la synchronisation. En particulier, un mécanisme de vues simplifié est nécessaire pour rediriger une méthode vers une autre méthode du même objet.

Notre travail se rapproche de celui de Jérôme Vouillon sur l'ajout d'un mécanisme de vues sophistiqué dans un langage à objets séquentiel, et en réutilise de nombreuses techniques, bien que différant dans ses motivations et aussi dans ses solutions.

6.1.3 Sous-typage

Participants : Didier Rémy, François Pottier.

Didier Rémy et François Pottier ont travaillé sur le sous-typage, et en particulier sur l'utilisation du mécanisme de sous-typage pour typer la concaténation des enregistrements, les types sommes, et les messages dynamiques.

François Pottier est, depuis le 1^{er} octobre 1999, chargé de recherche au sein du projet Cristal. Il a repris ses travaux sur l'inférence de types à base de contraintes de sous-typage, et se prépare à publier une bibliothèque de gestion de contraintes efficace et applicable à de nombreux problèmes distincts. L'étude théorique de ces problèmes, par exemple l'optimisation de code et la sécurité de l'information, constitue actuellement son principal thème de recherche.

6.1.4 Polymorphisme extensionnel et fonctions génériques

Participants : Pierre Weis, Jun Furuse.

En collaboration avec Jun Furuse, Pierre Weis a écrit un article sur un système sûr de lecture-écriture de valeurs Caml sur des fichiers externes. On entend ici par système sûr, un système qui ne provoque pas d'erreur à l'exécution, et qui permet donc un certain nombre de vérification de type à l'exécution. La solution retenue est basée sur le polymorphisme extensionnel, les valeurs dynamiques et un algorithme d'identification des définitions de types de données par clés MD5. Cet article a été présenté à la conférence JFLA'00.

Outre l'implémentation de ce système sûr d'entrées-sorties de valeurs, Jun Furuse a poursuivi l'implémentation du polymorphisme extensionnel dans le compilateur Objective Caml

2.02. Il a implémenté la vérification statique des types d'emploi des fonctions génériques; pour cela, on a changé l'algèbre de type du polymorphisme extensionnel, pour y intégrer les versions abstraites des fonctions génériques, qui sont maintenant directement lisibles dans les types. Ainsi, les types des fonctions génériques sont plus informatifs et la vérification statique s'intègre naturellement au typage et à la modularité du langage.

Avec Pierre Weis, Jun Furuse a étudié le problème de la «cohérence», c'est-à-dire le problème de l'inférence des utilisations surchargées de fonctions génériques. Cette inférence est liée à l'inférence des types recevables pour les variables de type dynamiques qui ne sont pas complètement instanciées dans les programmes, après l'inférence de types habituelle de ML. La solution proposée repose sur l'instanciation de ces variables par l'anti-unificateur des filtres acceptables des fonctions génériques auxquelles ces variables sont appliquées (parties «inévitables» des motifs qui définissent la fonction générique). Cette méthode permet, lorsque le programme n'est pas ambigu, d'inférer le type d'emploi des fonctions génériques et donc de règler le problème de cohérence. Dans les cas complètement ambigus, un message d'erreur est émis.

L'introduction des valeurs dynamiques a également permis de régler le problème des variables de type dynamiques non instanciées et libres dans l'environnement de typage: toutes ces occurrences de variables de type dynamiques libres sont maintenant unifiées au type `dyn` (le type des valeurs dynamiques, c'est-à-dire des valeurs accompagnées de leur type). Cette méthode est sûre et adéquate pour bon nombre d'exemples (penser à `print (eval (read ()))`), mais ses conséquences pour le polymorphisme extensionnel sont encore à explorer. Plus généralement, le lien entre polymorphisme extensionnel et valeurs dynamiques est peu clair et constitue une voie de recherche intéressante.

6.1.5 Typage polymorphe des définitions récursives

Participants : Pierre Weis, François Pessaux.

Le typage traditionnel des définitions récursives en ML impose que toutes les occurrences de l'identificateur défini récursivement soit du même type exactement. Cela empêche le typage de fonctions pourtant raisonnables, soit définies par récursion mutuelle avec une autre fonction (`let rec map = ... and map_succ l = map succ l;;`), soit définies sur des types de données *récursivement polymorphes*, c'est-à-dire comprenant des constructeurs dont l'argument a un type contenant une occurrence du type défini sous un autre constructeur de types.

Pour pallier cet inconvénient Mycroft a proposé une autre règle de typage qui autorise l'utilisation de l'identificateur défini récursivement avec un type différent du type de définition. Toutefois cette méthode, connue sous le nom de règle μ, a été prouvée indécidable et n'est donc pas raisonnable pour un compilateur.

En collaboration avec François Pessaux, Pierre Weis a terminé l'étude et l'implémentation d'une restriction décidable de cette règle. C'est une extension stricte de la règle habituelle de ML, qui est simple mais qui permet cependant de typer tous les programmes donnés dans la littérature comme des exemples de typage récursif polymorphe non typables en ML. En outre, notre méthode est efficace car elle n'impose pas le retypage itératif de toute la définition récursive comme l'algorithme original de Mycroft. Elle autorise aussi un typage incrémental des programmes analogue à l'algorithme habituel de Caml.

La nouvelle règle n'implémente évidemment pas la règle μ dans toute sa généralité, mais elle permet en particulier la définition polymorphe d'identificateurs dont la définition fait intervenir l'identificateur en cours de définition récursive. On ne connaît pas d'exemples de programme ML bien typé dont le type ne soit pas synthétisé par notre algorithme.

Cet algorithme a été implémenté dans l'analyseur d'exceptions que François Pessaux a développé pour sa thèse. Dans sa thèse, François Pessaux a prouvé formellement les propriétés de terminaison et de correction de cet algorithme.

6.1.6 Polymorphisme explicite

Participants : Jacques Garrigue, Didier Rémy.

Didier Rémy et Jacques Garrigue ont continué le travail sur le polymorphisme explicite [4], étudiant l'intégration de ce mécanisme au langage OCaml afin d'ajouter au langage des méthodes polymorphes. Ils ont également envisagé différentes extensions, notamment l'ajout de polymorphisme prénexe de rang supérieur qui permettrait de spécialiser des méthodes polymorphes dans des sous-classes.

6.1.7 Polymorphisme structurel

Participants : Jacques Garrigue, Didier Rémy.

Les systèmes de typage à polymorphisme structurel proposent une voie médiane entre le polymorphisme paramétrique, où un type polymorphe représente une boîte noire inaccessible pendant le calcul, et le polymorphisme de sous-typage, où l'on dispose d'une information partielle sous la forme d'un super-type permettant un certain nombre d'opérations, qui doivent être implémentées dans tous ses sous-types.

Dans le polymorphisme structurel on peut connaître un certains nombre d'informations structurelles sur un valeur de type polymorphe (une partie des champs disponibles dans un enregistrement, la présence ou l'absence de certains constructeurs dans un type somme) sans connaître son type exact. Deux applications sont le typage des objets en Objective Caml, et le typage des sommes polymorphes en Objective Label. En ce sens le polymorphisme structurel a un pouvoir expressif qui se rapproche du sous-typage, tout en se basant sur les mêmes principes que le polymorphisme paramétrique.

6.2 Analyses statiques

6.2.1 Détection statique de levée d'exceptions

Participants : Xavier Leroy, François Pessaux.

Ce travail, partiellement financé par le Centre National d'Études des Télécommunications, porte sur la réalisation d'un analyseur statique de programmes écrits en Objective Caml, permettant la détection des exceptions levées et non rattrapées pouvant mettre fin à l'exécution des-dits programmes.

L'aspect théorique de cet analyseur repose sur l'utilisation de techniques à base d'unification sur des termes de type cycliques à la ML ainsi que sur des termes de rangées. De plus, la précision de l'analyse doit beaucoup à l'ajout d'une opération de *soustraction* sur ces types. Cette opération permet de rompre l'accumulation de contraintes induite par l'unification, permettant ainsi l'apport de contraintes négatives sur les types. Contrairement à ce que l'utilisation d'un mécanisme d'unification pourrait laisser penser (flot bi-directionnel des contraintes, donc perte de précision), cette analyse peut se permettre d'être plus fine au niveau des arguments des exceptions et des structures de données que les autres analyses existantes, amenant ainsi à une meilleure précision globale que ces dernières qui se doivent de restreindre leur approximations afin de conserver des temps d'analyse raisonnables. Ainsi, l'analyseur semble montrer que l'utilisation de techniques «plus faibles» peut en pratique donner des résultats au moins aussi acceptables que d'autres analyses basées sur des cadres théoriques plus précis.

L'aspect pratique de ce travail est illustré par la réalisation d'une implantation «grandeur nature» d'un analyseur en Objective-Caml pour Objective-Caml. L'analyseur est encore en cours de développement, mais traite désormais le langage complet à l'exception des aspects objets (un prototype a été réalisé à part, montrant que l'extension du formalisme aux objets ne devrait pas poser de problème particulier, et attend d'être fusionné avec l'implantation courante). En particulier, la gestion de l'analyse séparée (par unité de compilation) a été gérée, ce qui ne pose pas de réels problèmes sur le plan théorique, mais relève d'un travail complexe d'ingénierie. De plus le système de modules du langage est dorénavant couvert par l'analyse, nécessitant à la fois un cadre théorique et la résolution de problèmes d'implantation pointus.

L'an passé François Pessaux avait continué à implanter son analyseur sur le langage Objective Caml tout entier et avait établi formellement les propriétés de son analyse.

Au cours de l'année 1999, il a continué ce travail d'implémentation pour obtenir une version finale et fonctionnelle de son logiciel d'analyse dont une distribution officielle est prévue dans les mois qui viennent. Une partie non négligeable de son temps a été également dévolue à la rédaction de son mémoire de thèse soutenue le 16 décembre 1999 à l'Université Paris 6.

6.2.2 Flots synchrones et programmation fonctionnelle

Participants : Michel Mauny, Pascal Cuoq.

Pascal Cuoq est doctorant au sein du projet Cristal depuis septembre 1998 sous la direction de Marc Pouzet (LIP6, Univ. Paris 6) et Michel Mauny. Ce travail, partiellement financé par le CNET, a pour objectif général l'amélioration du langage Lucid Synchrone.

Lucid Synchrone est un langage de flots synchrones, similaire à Lustre, mais incorporant de l'ordre supérieur. Pascal Cuoq a cette année mis au point une analyse statique vérifiant que les programmes Lucid Synchrone sont «causaux». Les programmes causaux sont ceux pour lesquels le calcul de la valeur d'une variable ne fait pas appel à la valeur instantanée (étant en train d'être calculée) de cette même variable. Pour prendre un exemple utilisant de l'ordre supérieur, cette analyse de causalité exprime le fait que la fonction Lucid Synchrone `fun f -> let rec x = f x in x` ne peut être appliquée qu'à une fonction dont la valeur renvoyée ne dépend pas instantanément de son argument.

Le problème suivant est de générer du code pour les nouveaux programmes acceptés par cette analyse: l'ordonnancement correct du code généré pour une fonction pouvant dépendre de la façon dont celle-ci est utilisée.

Précédemment, n'étaient acceptés dans Lucid Synchrone que des programmes pour lesquels on pouvait produire du code séquentiel d'une façon modulaire. Une autre solution existante (utilisée dans Lustre) est d'*inliner* systématiquement toutes les fonctions, mais cette solution n'est pas modulaire et peut produire un code de taille exponentielle par rapport au programme d'entrée.

L'objectif actuel est l'étude d'une troisième voie, dans laquelle une approche similaire à l'évaluation paresseuse est utilisée pour partager le code généré pour une fonction entre toutes ses applications (donc entre différents ordonnancements), tout en conservant les garanties de réactivité de Lucid Synchrone (en pratique, sans allocation de mémoire dynamique).

6.3 Implémentations de Caml

6.3.1 Objective Caml

Participants : Xavier Leroy, Jérôme Vouillon, Damien Doligez [projet Para], Luc Maranget [projet Para].

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).

Cette année, nous avons publié les versions 2.02, 2.03 et 2.04 d'Objective Caml. Ces versions ne présentent pas de modifications majeures du langage, mais consolident l'implémentation du nouveau langage de classes introduit dans la version 2.00, et améliorent le confort d'utilisation du système sur de nombreux points: diagnostics d'erreurs et d'avertissements plus précis, enrichissement de la bibliothèque standard, ...

Xavier Leroy a implémenté plusieurs optimisations nouvelles dans le compilateur: combinaison des allocations d'objets en mémoire situées dans le même bloc de base en une seule allocation; construction statique des fermetures de fonctions sans variables libres; amélioration de la propagation des constantes entières et de la gestion des bits de *tags*. Il a également étendu les possibilités d'interface avec d'autres langages, réalisé un portage sur le système BeOS et assuré la maintenance de l'ensemble du système.

Xavier Leroy et Pierre Weis ont revu, augmenté et corrigé leur livre «Le langage Caml». La nouvelle édition est parue en juillet 1999.

6.3.2 L'interface CamlIDL

Participant : Xavier Leroy.

Dans le but de faciliter l'interfaçage entre Objective Caml et d'autres langages de programmation, Xavier Leroy a réalisé CamlIDL, un générateur de code d'interface conforme au

modèle COM de Microsoft. 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. CamlIDL permet également d'échanger des objets et des classes entre Caml et C++.

Le code d'interface produit par CamlIDL est conforme à la norme COM de Microsoft, permettant ainsi d'utiliser des composants COM depuis une application Caml, mais aussi d'encapsuler du code Caml sous forme de composant COM, utilisable depuis n'importe quel langage disposant d'une interface COM (C, C++, Java, Visual Basic, etc).

CamlIDL facilite considérablement l'interopérabilité entre Caml et les autres langages de programmation, et permet en particulier d'écrire en Caml les parties les plus algorithmiquement difficiles d'une application, les autres parties (interface utilisateur, programme principal, etc) étant écrites dans un langage imposé (C++, Visual Basic).

6.3.3 OCaml et tableaux Fortran

Participants : Xavier Leroy, Manuel Serrano, Damien Doligez [projet Para].

Le CEA et plus particulièrement la DRN (Division des Réacteurs Nucléaires) utilise OCaml comme langage de script. Dans ce cadre, OCaml est utilisé comme langage de composition de programmes Fortran. Divers calculs scientifiques implantés en Fortran77 sont enchaînés les uns après les autres. Un programme central implanté lui en OCaml joue alors le rôle de «moteur». C'est lui qui décide l'ordre d'enchaînement des calculs Fortran.

OCaml dispose d'ores et déjà de facilités d'interfaçage avec des codes externes : il est ainsi possible d'interfacer OCaml et Fortran77 par le biais de l'interface avec le langage C, et d'un langage de description d'interfaces (CamlIDL) à même de générer automatiquement des *stubs* pour l'interface avec C. Cependant, les mécanismes existants souffrent d'un certain nombre de limitations et on ne dispose d'aucun outil spécifique à l'interfaçage avec Fortran77. En particulier le type de traitements opérés par la DRN fait un usage intensif du calcul matriciel. Souvent il s'agit de construire des matrices réelles Fortran puis de leur faire subir plusieurs traitements consécutifs. Puisque c'est OCaml qui a la charge de déterminer l'enchaînement des traitements il est important qu'il puisse manipuler ces matrices. Ceci n'est que très partiellement possible dans la version actuelle d'OCaml.

Manuel Serrano, collaborateur extérieur du projet Cristal, a travaillé sur une extension de OCaml qui comble ce manque. Il a travaillé à l'élaboration d'une interface de programmation permettant l'utilisation de matrices Fortran en OCaml ainsi qu'à son implantation. Grâce à cette extension, OCaml peut dorénavant récupérer des matrices construites dans des modules Fortran, éventuellement les modifier, les archiver sur disque puis les relire, et enfin, les retourner à Fortran.

6.3.4 Interopérabilité

Participants : Xavier Leroy, Bruno Pagano, Luc Maranget [projet Para].

Dans le cadre d'un contrat avec la société Microsoft, Xavier Leroy, Bruno Pagano et Luc

Maranget travaillent sur l'intéropérabilité entre langages de programmation. Ce travail est actuellement couvert par un accord de confidentialité.

6.3.5 Intégration des compilateurs Objective Caml et Objective Label

Participant : Jacques Garrigue.

Le langage Objective Label, développé par Jacques Garrigue à l'Université de Kyoto, étend Objective Caml dans trois directions:

- ajout de paramètres étiquetés et optionnels,
- ajout d'un nouveau type de sommes polymorphes,
- possibilité d'avoir du polymorphisme à l'intérieur des types de méthodes.

Le but de l'intégration était de fournir ces différentes extensions dans la version standard du compilateur Objective Caml, tout en tenant compte des impératifs de simplicité et de correction théorique.

Jacques Garrigue, en coopération avec les autres membres du projet Cristal, a réalisé cette intégration, tout en approfondissant ses bases théoriques.

6.3.6 Le préprocesseur Camlp4

Participant : Daniel de Rauglaudre.

Daniel de Rauglaudre a amélioré son logiciel Camlp4, préprocesseur pour Objective Caml.

Le type des lexèmes (tokens) est devenu plus général et les analyseurs lexicaux ont maintenant toute latitude pour définir les types des valeurs qu'ils renvoient, y compris la notion de mot-clé. La cohérence reste assurée dans les grammaires, grâce à une fonction de l'analyseur lexical destinée à vérifier chaque constructeur utilisé.

Un programme Lablp4 a été ajouté pour la version Olabl de Jacques Garrigue, version d'OCaml étendue avec, entre autres, des étiquettes (labels) pour distinguer les paramètres des fonctions.

Camlp4 suit de très près les évolutions d'OCaml, quand des constructions syntaxiques sont ajoutées ou modifiées, en particulier dans les aspects orientés-objet du langage.

6.3.7 Évaluation Partielle

Participant : Pierre Weis.

En collaboration avec Dick Kieburtz de l'Oregon Graduate Institute, Pierre Weis a continué l'implémentation d'un évaluateur partiel pour Caml Light. Cet évaluateur partiel travaille sur le code lambda et génère du code lambda et l'évaluation partielle «passe la frontière des modules»: les formes partiellement évaluées des fonctions et des expressions sont disponibles d'un module à l'autre et le compilateur peut les utiliser. Cet évaluateur partiel ne traite pas les effets de bord qu'il laisse inchangés. Cela pose un gros problème pour l'évaluation partielle des structures de

données complètement fonctionnelles puisque le compilateur de Caml Light génère des effets de bord pour leur gestion, et ces effets de bord «fonctionnels» sont indissociables des effets de bord de l'utilisateur; en conséquence l'évaluateur partiel est limité à l'évaluation des fonctions pures. La solution de ce problème impliquerait la modification du compilateur pour qu'il conserve l'information de pureté des structures de données.

Une autre voie de recherche a été de chercher un ensemble de programmes qui permettraient de tester l'efficacité de l'évaluation partielle. À cette occasion, Pierre Weis a commencé la traduction en Objective Caml des exemples distribués avec Caml Light et du programme Micro Hawks de simulation de circuits, développé en Haskell à l'OGI.

6.3.8 Concours ICFP

Participants : Pascal Cuoq, Damien Doligez [projet Para], Xavier Leroy, Luc Maranget [projet Para], Alan Schmitt [projet Para].

Un programme écrit par Pascal Cuoq, Damien Doligez (Para), Xavier Leroy, Luc Maranget (Para) et Alan Schmitt (Para) a obtenu le premier prix au concours de programmation organisé à l'occasion du congrès ICFP 1999.

6.4 Applications OCaml

6.4.1 Bibliothèque de traitement d'images

Participants : François Pessaux, Pierre Weis, Jun Furuse.

Initiée par François Pessaux et Pierre Weis, la bibliothèque Camlimages ne traitait à l'origine que le format d'image BMP. En collaboration avec Pierre Weis, Jun Furuse a grandement étendu et généralisé cette bibliothèque, y ajoutant le traitement de nombreux autres formats d'images, ainsi que de nombreux algorithmes de traitement d'images, en particulier la possibilité de manipuler de très grosses images qui ne tiennent pas en mémoire centrale.

Cette bibliothèque va être distribuée dans le recueil des bibliothèques d'Objective Caml, le «bazar OCaml».

Serveur Web de bases de données généalogiques

Participant : Daniel de Rauglaudre.

Daniel de Rauglaudre a continué à développer GeneWeb, logiciel de gestion de bases de données généalogiques, dont l'interface se fait via un navigateur Web, utilisable soit localement sans connexion internet, soit sur le réseau en serveur HTTP ou en CGI, un même serveur pouvant servir plusieurs bases de données.

De nombreuses améliorations ont été apportées pour rendre le programme plus convivial et plus complet.

Un programme nommé «gwsetup» a été ajouté pour pouvoir créer et gérer des bases de données sans avoir à taper des commandes dans des langages de style *shell* (`sh` ou `dos`): ce programme fonctionne de la même façon, en serveur HTTP, avec interface par navigateur Web.

Une collaboration avec des utilisateurs de différents pays ont permis d'ajouter cinq langues: le tchèque (avec résolution du problème des déclinaisons), le danois, l'hébreu, l'islandais et le norvégien. GeneWeb «parle» maintenant 15 langues au total.

L'accès privé aux données utilise maintenant en mode serveur (pas CGI) le protocole HTTP pour l'entrée des mots de passe. D'autres types d'accès protégés ont été ajoutés et il est possible de définir des listes d'utilisateurs pour chacun de ces accès.

La documentation en HTML a été complétée. Le programme permet maintenant d'accéder directement à cette documentation.

Une collaboration avec une utilisatrice a permis d'améliorer notablement la présentation des pages, avec conception d'une icône spécifique pour le logiciel.

GeneWeb est un logiciel libre et gratuit, avec licence GPL et sources distribuées. Il est écrit en Objective Caml et utilise le préprocesseur Camlp4. Les distributions comportent une version Windows, une version Unix, une version paquetage RedHat de Linux, ainsi que les sources.

Toutes les informations sur GeneWeb sont à l'adresse `http://pauillac.inria.fr/~ddr/GeneWeb/`.

6.5 Arithmétique

Participants : Robert Harley, Michel Mauny, Daniel de Rauglaudre.

Dans le cadre de son travail de thèse Robert Harley a porté son attention sur l'arithmétique rapide en caractéristique 2, ce qui vient complémenter l'arithmétique des grands entiers. Un résultat particulier est l'implémentation d'une multiplication très efficace de polynômes avec des coefficients booléens, utilisant des transformées de Fourier additives. Un autre est l'optimisation des opérations sur des petites quantités.

Ce dernier a rendu possible le calcul d'un logarithme discret sur une courbe elliptique définie sur GF(2^{97}), le plus difficile jamais réalisé, avec l'assistance de 195 internautes. Ce résultat a fait l'objet de plusieurs communiqués de presse et a été relaté dans Le Monde, le «Irish Times», le «Toronto Globe & Mail» parmi d'autres. À la fin de l'année, Robert Harley s'attaque au défi suivant (ECC2K-108), aidé en cela par Daniel de Rauglaudre qui a réalisé un certain nombre d'outils (Internet, notamment) permettant de gérer l'intéraction avec le grand nombre de participants attendus.

Par ailleurs Robert Harley a intégré son implémentation des algorithmes de Toom et de Schönhage à la bibliothèque GMP avec l'aide de son auteur, Torbjörn Granlund. Cette implémentation permet d'accélérer la multiplication des très grands entiers et sera disponible dans la prochaine version de GMP, dont la distribution est prévue pour le début de l'an 2000.

7 Contrats industriels (nationaux, européens et internationaux)

7.1 Action de recherche coordonnée Java Card

Nous participons à l'action de recherche coordonnée Java Card `http://www.irisa.fr/lande/jensen/javacard.html`

7.2 CNET: Analyse d'exceptions pour Objective Caml

Le travail de François Pessaux sur l'analyse d'exceptions de programmes Objective Caml est partiellement financé par le CNET.

7.3 CNET: Flots synchrones en ML

Le travail de Pascal Cuoq sur l'adjonction de synchronisme dans les langages fonctionnels est partiellement financé par le CNET.

7.4 Microsoft: interopérabilité

Ce contrat, avec la société Microsoft, porte sur l'interopérabilité et est actuellement couvert par un accord de confidentialité. Il finance le travail de Bruno Pagano, et Xavier Leroy en est le responsable (*principal investigator*).

8 Actions régionales, nationales et internationales

8.1 Actions nationales

Michel Mauny est membre de l'équipe de direction du GDR 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.

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

8.3 Relations bilatérales internationales

8.3.1 Amérique du Nord

Pierre Weis a passé 2 mois à l'Oregon Graduate Institute dans le laboratoire du professeur Dick Kieburtz, où il a continué l'étude de l'évaluation partielle dans le compilateur Caml Light.

9 Diffusion de résultats

9.1 Mobilité

Xavier Leroy a passé six mois en disponibilité dans la société Trusted Logic, où il a travaillé sur la vérification de *bytecode* JavaCard embarquée sur cartes à puces et autres systèmes disposant de faibles ressources. De retour à l'INRIA, il continue sa collaboration avec Trusted Logic en tant que consultant.

Pierre Weis a passé deux mois à l'Oregon Graduate Institute (Portland, USA) où il a travaillé sur l'évaluation partielle dans le compilateur Caml-Light.

9.2 Animation de la communauté scientifique

Michel Mauny est vice-président du Comité des Projets de l'INRIA-Rocquencourt. Il est membre suppléant élu de la commission d'évaluation de l'INRIA. Il a été membre du jury d'admission du concours CR INRIA en 1999, et a fait partie des sections locales d'audition de ce concours pour les Unités de Recherche de Rocquencourt, Rennes et Sophia-Antipolis. Il fait partie de la Commission de Spécialistes de l'université de Rouen en tant que suppléant.

Xavier Leroy est membre titulaire élu de la commission d'évaluation de l'INRIA.

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

Michel Mauny a été responsable scientifique (et a donc constitué le programme) du cycle intitulé «Les réseaux informatiques: conception et programmation», dans le cadre plus général d'une série de conférences associée à l'exposition «Nouvelles images, nouveaux réseaux» de la Cité des Sciences et de l'Industrie, à La Villette, qui a eu lieu au début de l'année 1999.

Depuis mai 1999, Pierre Weis est co-responsable (avec Michel Loyer) des Moyens Informatiques de Rocquencourt. Au sein de ce tandem, Michel Loyer est plus particulièrement chargé de la gestion administrative des commandes tandis que Pierre Weis s'occupe des bonnes relations avec les chercheurs, de l'écoute de leurs besoins et facilite le passage de commandes.

Pierre Weis est membre du Comité d'UR de Rocquencourt.

Pierre Weis gère et modère la tribune de discussion de Caml (540 abonnés). En outre, Pierre Weis continue à suivre et à encourager les efforts des professeurs des classes préparatoires. Il s'est rendu à Toulouse au colloque de l'association des professeurs de mathématiques des classes préparatoires, du 2 au 5 mai 1999, où il a fait un cours d'introduction aux modules et aux objets en Objective Caml.

9.2.1 Comités de lecture et programmes

Didier Rémy a été responsable (*general chair*) du colloque PLI (*Principles, Logics, and Implementations of high-level languages*) organisé par l'INRIA en collaboration avec l'ACM-SIGPLAN qui s'est tenu à Paris du 17 septembre au 1er octobre. Ce colloque a regroupé les conférences ICFP (*International Conference on Functional Programming*) et PPDP (*Principles and Practice of Declarative Programming*) et 9 séminaires (voir `http://pauillac.inria.fr/pli`). Le colloque a réuni environ 280 chercheurs internationaux.

Didier Rémy est membre du comité de direction de la conférence ICFP (*International Conference on Functional Programming*) qu'il a présidé jusqu'en septembre 1999. Il a été

responsable de l'organisation de cette conférence en 1999 qui s'est tenue à Paris durant le colloque PLI.

Didier Rémy est coéditeur avec Kim Bruce (Williams College, MA) d'un numéro spécial de la revue TAPOS (*Theory and Practice of Object-oriented Systems*) dédié aux meilleurs travaux présentés au séminaire FOOL 5.

Xavier Leroy a fait partie du comité de programme des congrès Principles and Practice of Declarative Programming 1999, European Symposium on Programming 2000, et Programming Language Design and Implementation 2000.

Xavier Leroy a organisé la réunion de septembre 1999 du groupe de travail IFIP 2.8 – programmation fonctionnelle –, qui s'est tenue en septembre à Saint-Malo.

Pierre Weis a été président du comité de programme des JFLA 99, qui se sont déroulées début février 1999.

Michel Mauny est membre du comité de lecture de la revue Techniques et Sciences Informatiques.

9.2.2 Autres activités d'intérêt général

Diffusion des connaissances

Participant : Pierre Weis.

Pierre Weis a réalisé une troisième version du CD-ROM de diffusion des logiciels de l'INRIA, «Logiciels libres à l'INRIA Rocquencourt». 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 1300 exemplaires cette année. La quatrième version de ce CD-ROM est en cours de réalisation et devrait sortir au début du mois de janvier 2000.

En collaboration avec Marie-Pierre Durollet et Annie Garot, Pierre Weis a réalisé deux CD-ROMs de diffusion des rapports de recherches de l'INRIA: le premier CD-ROM contient tous les rapports de techniques et de recherche depuis la fondation de l'INRIA jusqu'à 1997. Le deuxième contient les rapports d'activités et les rapports techniques et de recherche jusqu'à janvier 1999. Ces CD-ROMs multi-plateformes ont été diffusés à plusieurs centaines d'exemplaires cette année.

Configurations matériels/logiciels

Participants : Michel Mauny, Louis Audoire [service Amibe].

Michel Mauny effectue l'installation et la configuration du système d'exploitation Linux sur un disque «maître» de portable Dell, et a réalisé un programme de duplication de cette matrice (installant aussi Linux ou bien Windows, ou alors les deux à la fois). Ainsi, le service Amibe peut aisément dupliquer cette installation sur les portables Dell des utilisateurs qui en font la demande.

9.3 Enseignement

9.3.1 Encadrement et jurys

Xavier Leroy encadre la thèse de François Pessaux (avec Christian Queinnec, Univ. Paris 6) et co-encadre (avec Benjamin Werner, projet COQ) la thèse de Benjamin Grégoire. Ils ont aussi co-encadré son stage de DEA (Univ. Paris 7).

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

Didier Rémy encadre la thèse de Jérôme Vouillon.

Pierre Weis encadre la thèse de Jun Furuse.

9.3.2 Enseignements de troisième cycle universitaire

Xavier Leroy assure le cours Typage et Programmation (20h) dans le DEA «Sémantique, Preuves et Programmation» (SPP, paris 6-7-11, X, ENS)).

Michel Mauny est Professeur Associé à temps partiel au Pôle Universitaire Léonard de Vinci (PULV). Il y a dispensé un cours de programmation en Java (25h) dans une formation de niveau 3ième cycle du style DESS. Il est aussi membre de l'équipe de direction du DEA SPP.

9.3.3 Enseignement en écoles d'ingénieurs

Michel Mauny a donné une semaine de cours/TD (15h) de Programmation Fonctionnelle à l'ISIA (Sophia-Antipolis), et un cours de compilation au Pôle Universitaire Léonard de Vinci (25h).

Didier Rémy est Maître de Conférence à temps partiel à l'École Polytechnique. A ce titre il a enseigné un cours de compilation et un cours sur la modularité.

Jusqu'à septembre 1999, Pierre Weis a assuré 4 heures hebdomadaires de travaux dirigés à l'École Polytechnique, en temps que Chef de Travaux Pratiques dans le cadre du tronc commun d'informatique dirigé par Jean-Jacques Lévy et Robert Cori.

Bruno Pagano est Maître de Conférences vacataire à l'École Nationale Supérieure des Techniques Avancées (ENSTA). Il y donne 20 heures de travaux dirigés dans un cours de programmation en C.

Robert Harley participé à l'enseignement du cours de Compilation en Majeure 2 de l'École Polytechnique, en participant aux travaux dirigés, deux heures par semaine.

François Pessaux a assuré l'encadrement des séances d'initiation à l'informatique des élèves de Tronc Commun d'informatique de l'École Polytechnique durant le mois de juin 1999 (20h). Il assure aussi l'encadrement des travaux dirigés du Tronc Commun d'informatique des élèves de l'École Polytechnique pour l'année scolaire en cours. Ce travail d'enseignement, effectué en compagnie de Luc Maranget et Robert Cori a débuté en septembre de cette année et se poursuivra jusqu'en février 2000, pour un total de 44 heures.

9.3.4 Enseignements de premier et second cycles universitaires

Pascal Cuoq et François Pessaux assurent l'encadrement des travaux pratiques de Licence d'informatique à l'Université de Paris 6 (48h chacun).

9.3.5 Autres enseignements

Pierre Weis a donné un cours d'approfondissement à Caml Light aux professeurs de l'Union des professeurs de Mathématiques Spéciales (16 heures).

9.4 Participation à des colloques, séminaires, invitations

Xavier Leroy et François Pessaux ont participé au congrès Principles of Programming Languages (San Antonio, janvier 1999), et François Pessaux y a présenté son travail sur la détection d'exceptions [7].

Xavier Leroy a participé à la réunion du groupe de travail IFIP 2.8, qu'il a organisée et qui s'est déroulée en septembre 1999 à Saint-Malo. Il y a présenté son travail sur l'interface CamlIDL.

Jun Furuse, Xavier Leroy, Michel Mauny, Francois Pessaux, François Pottier et Didier Rémy ont assisté au groupe de conférences PLI *Principles, Logics, and Implementations of high-level languages*), incluant notamment les conférences ICFP (*International Conference on Functional Programming*), PPDP (*Principles and Practice of Declarative Programming*) ainsi que le séminaire HOOTS (*Higher-Order Operational Techniques for Semantics*).

Xavier Leroy a présenté à ICFP un exposé invité comparant les objets, les classes et les modules en Objective Caml, et a participé au congrès *Principles and Practice of Declarative Programming*, associé à ICFP.

Xavier Leroy a assisté au congrès Programming Languages Design and Implementation (Atlanta, mai 1999).

Michel Mauny, François Pottier et Michel Mauny ont participé à la réunion annuelle du groupe de travail APPSEM qui a eu lieu à l'université d'Edimbourg du 7 au 9 septembre. François Pottier y a présenté ses travaux les plus récents, et Didier Rémy y a présenté ses travaux avec Cédric Fournet (projet Para), Cosimo Laneve (Université de Bologne) et Luc Maranget (projet Para) sur la conception d'un langage à objets avec de la concurrence intime et l'héritage simultané des comportements et de la synchronisation.

Didier Rémy a participé au séminaire CONFER qui s'est tenu à Paris en Novembre 99.

François Pottier et Didier Rémy ont été invités à faire chacun une présentation au colloque inaugural du laboratoire PPS de l'Université Paris 7, qui a eu lieu à l'École Normale du 4 au 6 octobre.

Pascal Cuoq et Pierre Weis ont participé aux Journées Francophones des Langages Applicatifs (JFLA) qui se sont déroulées du 31 janvier au 2 février 1999.

Robert Harley a participé à l'École de Printemps d'Informatique Théorique sur le codage et la cryptographie, qui s'est tenue à Batz-sur-Mer en mai 1999.

Pascal Cuoq a participé au séminaire «Synchrone» qui s'est tenu à Hyères en novembre 1999.

Michel Mauny a participé à la réunion du groupe de normalisation ISO du langage ASN.1 qui s'est tenue à Lannion en janvier 1999.

Bruno Pagano s'est rendu à Redmond (USA) pour participer à un groupe de travail organisé par la société Microsoft.

Xavier Leroy, Michel Mauny et Benjamin Grégoire ont participé à la réunion de l'Action de Recherche Coopérative JavaCard qui a eu lieu à Saint Malo en juillet.

10 Bibliographie

Livres et monographies

[1] E. Chailloux, P. Manoury, B. Pagano, *Objective Caml: developpement d'applications*, O'Reilly, 2000, à paraître.

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

Thèses et habilitations à diriger des recherches

[3] F. Pessaux, *Détection statique d'exceptions non rattrappées en Objective Caml*, thèse de doctorat, Univ. Paris 6, 1999.

Articles et chapitres de livre

[4] J. Garrigue, D. Rémy, « Extending ML with Semi-Explicit Higher-Order Polymorphism », *Theory And Practice of Objects Systems*, 1999, à paraître. Une version préliminaire est parue à TACS'97.

[5] X. Leroy, F. Rouaix, «Security properties of typed applets», *in : Secure Internet Programming – Security issues for Mobile and Distributed Objects*, J. Vitek et C. Jensen (éditeurs), *Lecture Notes in Computer Science*, *1603*, Springer-Verlag, 1999, p. 147–182, `http://pauillac.inria.fr/~xleroy/publi/sip-typed-applets.ps.gz`.

[6] X. Leroy, « A modular module system », *Journal of Functional Programming*, novembre 1999, accepté pour publication, à paraître.

Communications à des congrès, colloques, etc.

[7] F. Pessaux, X. Leroy, « Type-based analysis of uncaught exceptions », *in : Proceedings 26th ACM symposium Principles of Programming Languages*, ACM Press, p. 276–290, janvier 1999, `http://pauillac.inria.fr/~xleroy/publi/exceptions-popl.letter.ps.gz`.

Rapports de recherche et publications internes

[8] X. Leroy, D. Rémy, J. Vouillon, D. Doligez, *The Objective Caml system, documentation and user's manual – release 2.03*, INRIA, novembre 1999, documentation distribuée avec le système Objective Caml, `http://caml.inria.fr/ocaml/htmlman/`.

Divers

[9] B. Grégoire, *Certification en Coq de propriétés de sécurité issues du typage*, Rapport de dea, Université Paris 7, 1999.

[10] X. Leroy, F. Pessaux, « Type-based analysis of uncaught exceptions », février 1999, Article soumis aux ACM Transactions on Programming Languages and Systems.

[11] D. Rémy, J. Vouillon, « On the (un)reality of virtual types », a paraître en rapport de recherche, 1999.

[12] J. Vouillon, « An object calculus with views », juillet 1999, Unpublished note available electronically, `http://cristal.inria.fr/~vouillon/publi/views.ps.gz`.

[13] P. Weis, M.-P. Durollet, « Rapports techniques et de recherche 80-97 », INRIA, mars 1999, CD-ROM.

[14] P. Weis, M.-P. Durollet, « Rapports techniques et de recherche 97-98 », INRIA, mars 1999, CD-ROM.

[15] P. Weis, « Logiciels libres À l'INRIA Rocquencourt », INRIA, février 1999, CD-ROM.