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

Pierre Weis [DR, INRIA]

Responsable permanent

Didier Rémy [DR, INRIA]

Assistante de projet (commun avec COQ)

Nelly Maloisel

Personnel INRIA

Gérard Huet [DR]

Xavier Leroy [DR]

Michel Mauny [DR]

François Pottier [CR]

Daniel de Rauglaudre [IR]

Didier Rémy [DR]

Bruno Verlyck [IR, temps partiel]

Collaborateurs extérieurs

Guy Cousineau [professeur, université Paris 7]

Christian Queinnec [professeur, université Paris 6]

Manuel Serrano [maître de conférences, université de Nice]

Chercheurs invités

Vassily Litvinov [université de Washington, USA, d'octobre à fin décembre 2000]

Richard Kieburtz [Oregon Graduate Institute, de mars à fin juin 2000]

Ingénieur expert

Bruno Pagano [jusqu'à août 2000]

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 MNRT, université Paris 7, commun avec COQ]

Robert Harley [Allocataire MENRT, université Paris 7]

Tom Hirschowitz [AMN, université Paris 7]

Didier Le Botlan [AMX, université Paris 7]

Stagiaires

Vincent Simonet [Stagiaire de DEA, université Paris 7]

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 la cohérence des données manipulées 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 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 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 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 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 imposée par 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 explicitement typés. Citons les programmes Java qui nécessitent l'utilisation des «types virtuels» et qui sont rejetés par le langage alors qu'ils sont acceptés sans problème par OCaml. Ce travail fondamental pourrait, en ce sens, servir de base à la formalisation des types virtuels de Java.

Une autre approche est le sous-typage implicite qui a été étudié par François Pottier dans sa thèse. Le sous-typage explicite 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).

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.

[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.

3.1.3 Typage et analyses statiques

Nous abordons aussi l'analyse et la compilation optimisante de programmes par le biais de techniques de typage. C'est le cas de l'analyse d'exceptions, qui fournit dès la compilation des informations précises sur les exceptions susceptibles de se déclencher pendant l'exécution des programmes. François Pessaux a soutenu sa thèse sur ce sujet, et distribue cette année un analyseur d'exceptions complet pour le langage OCaml. De même, les informations fournies par le typage statique autorise la spécialisation du code à la compilation et l'analyse fine de la représentation des données, ce qui permet de produire du code efficace. Xavier Leroy a développé des techniques de représentations efficaces des 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 de 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. Le travail de Xavier Leroy et François Rouaix a constitué une première approche qui a ouvert la voie vers des techniques formelles de développement et de validation d'environnements d'exécution sûrs pour les *applets* d'un langage fortement typé. C'est le sujet de la thèse de Benjamin Grégoire, co-encadrée par Xavier Leroy et Benjamin Werner (projet Coq).

3.1.5 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.

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, presque 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.

Censément moderne, le style **à objets** se base 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'inspiration 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 fait 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 3.1.2). 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 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

[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.

[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.

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 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.

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, Didier Le Botlan.

Didier Le Botlan est doctorant au sein du projet Cristal depuis le 1^{er} septembre 2000. Encadré par Didier Rémy, il continue l'étude du typage des classes et des objets faite par Didier Rémy puis Jérôme Vouillon, en s'intéressant aux problèmes des définitions incrémentales de classes ou d'objets, des méthodes binaires, et de l'ordre supérieur sous forme de polymorphisme semi-explicite.

6.1.2 Polymorphisme explicite

Participants : Jacques Garrigue, Didier Rémy, Didier Le Botlan.

L'extension de ML proposée par Didier Rémy et Jacques Garrigue permet le polymorphisme d'ordre supérieur [GR99] et autorise la présence de méthodes polymorphes dans les objets.

Poursuivant le travail commencé avec Jacques Garrigue sur le polymorphisme explicite [GR99], Didier Le Botlan et Didier Rémy ont exploré l'ajout d'opérateurs de type d'ordre supérieur. Les opérateurs de type se sont révélés indispensables dès l'introduction des méthodes polymorphes (décrite dans [GR99]). Pour préserver l'extensibilité dans les classes paramétriques

[GR99] J. Garrigue, D. Rémy, « Extending ML with Semi-Explicit Higher-Order Polymorphism », *Journal of Functional Programming 155*, 1/2, 1999, p. 134–169, `ftp://ftp.inria.fr/INRIA/Projects/cristal/Didier.Remy/iandc.ps.gz`.

avec méthodes polymorphes, on est amené à remplacer, parmi les paramètres de la classe, des variables de rangée par des opérateurs de rangée qui sont appliqués à une variable polymorphe dans le type des méthodes polymorphes. C'est notamment le cas dans un schéma de programmation fréquent, connu sous le nom de schéma du *visiteur*. Jusqu'à présent de telles classes ne pouvaient être étendues, perdant ainsi l'intérêt principal de l'approche en style objet.

Didier Le Botlan étudie dans ce cadre une formalisation incluant à la fois le polymorphisme semi-explicite, l'ordre supérieur sur les types et un calcul d'objets. Un programme prototype permettant le typage et l'évaluation de programmes de test écrits dans le style OCaml a été implémenté et permet d'illustrer le travail théorique.

6.1.3 Typage par contraintes du calcul Join

Participants : Sylvain Conchon [projet Moscova], François Pottier.

Cousin du π-calcul, le calcul Join est un calcul de processus qui constitue une base formelle pour l'étude des langages de programmation concurrents et distribués. On sait qu'il est possible de doter le calcul Join d'un système de types proche de celui de ML; un tel système a d'ailleurs été implémenté dans JoCaml.

Cependant, pour réaliser des analyses de programmes poussées, il faut disposer de systèmes de types encore plus précis. Par exemple, le système de types de ML a été généralisé par Odersky, Sulzmann et Wehr pour produire le système *HM(X)* (système Hindley-Milner étendu par un système de contraintes X). HM(X) est paramétré par un langage de contraintes arbitraire qui permet de définir toute une famille de variantes de HM(X) (à base de contraintes de sous-typage, contraintes conditionnelles, etc.) sans nouvelle obligation de preuve.

François Pottier et Sylvain Conchon ont réalisé un programme similaire pour le calcul Join en proposant un système de types, nommé JOIN(X), lui aussi paramétré par un langage de contraintes arbitraire. On applique ainsi au calcul Join les techniques de résolution de contraintes développées pour d'autres langages. L'emploi de contraintes de sous-typage, en particulier, sera indispensable à l'analyse de flots d'information pour le calcul Join. Un article décrivant ces travaux a été soumis à la conférence ESOP'01.

6.1.4 Objets et concurrence

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

En collaboration avec Cédric Fournet (Microsoft Research, Cambridge), Cosimo Laneve (université de Bologne) et Luc Maranget (projet Moscova), Didier Rémy a poursuivi ses travaux sur l'adjonction au calcul Join d'objets et de classes primitifs. Ces travaux ont abouti à la définition d'une extension à objets et à classes du calcul Join.

Ce calcul repose d'abord sur un transfert à un langage concurrent de la technologie des langages à objets et à classes et de leur typage, telle que celle développée pour le langage Ocaml. Cependant le contexte concurrent requiert de nouvelles constructions qui n'ont pas de contrepartie dans le cas séquentiel. Par exemple, la construction d'héritage appelée *raffinement*

sélectif permet d'hériter des clauses des classes parentes tout en modifiant leur synchronisation (étendu à l'héritage sélectif des clauses par un mécanisme de filtrage sur leur motif de synchronisation dans la classe parente). Cette construction est très expressive et permet de résoudre élégamment des problèmes difficiles souvent mentionnés dans la littérature sous le nom d'*anomalies d'héritage*.

Ce langage a été entièrement formalisé et la correction du typage par rapport à l'évaluation à été prouvée. Une partie 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 à New Delhi (Inde) [8].

6.1.5 Types polymorphes contraints

Participant : Daniel Bonniot.

Daniel Bonniot a intégré l'équipe Cristal au mois d'Octobre 2000. Il continue sa thèse *Extension et mise en oeuvre des systèmes de types polymorphes contraints* commencée à l'Ecole des Mines de Paris sous la direction de François Bourdoncle, qui reste son directeur de thèse. Il cherche actuellement à faire une présentation étendue du système de type $ML_{\leq}$ plus simple et plus proche des techniques habituelles.

Les types de $ML_{\leq}$ sont des types polymorphes contraints, de la forme $\forall V : K.\theta$, où V est un ensemble de variables de types, θ est un type monomorphe, et K une contrainte sur V. L'approche suivie est de présenter dans une première étape des règles de typage et d'inférence de type sur le lambda calcul non typé avec opérateurs, et de montrer la correction de ce typage. Dans une deuxième étape on étendra ce langage noyau. Les extensions prévues incluent la notion de classe – éventuellement paramétrique – (hiérarchie de constructeurs de types contenant des données), de méthode (fonction qui sélectionne une implémentation en fonction du type effectif de la classe (le *tag runtime*) de ses arguments), et les multi-méthodes, dont les méthodes à la Java ou C++ sont des cas particuliers.

Le langage de contraintes est lui aussi extensible, ce qui permettra notamment d'intégrer les travaux précédents de Daniel Bonniot sur la notion d'interface, pour typer plus précisément les méthodes usuelles.

Le système est mis en œuvre par l'implémentation d'un compilateur pour le langage Nice.

6.1.6 Multi-méthodes

Participants : Didier Rémy, François Pottier, Daniel Bonniot, Vassily Litvinov.

Vassily Litvinov a effectué un stage de 3 mois dans le projet Cristal encadré par Didier Rémy. Ce stage a permis de réfléchir à une extension de ML avec une forme minimale de multi-méthodes à la Cecil. Le but est d'explorer une alternative à l'approche à enregistrements sur laquelle repose actuellement le modèle des objets du langage Ocaml, et de voir s'il est envisageable d'enrichir la couche à objets de Ocaml par une approche mixte à enregistrements et à surcharge. Ces travaux préliminaires ont avant tout permis de mieux décomposer le problème du typage en une première étape, classique, de génération de contraintes, paramétrée par un langage de contraintes adapté aux multi-méthodes, suivie d'une seconde étape de résolution des

contraintes qui autoriserait différentes variantes selon la puissance du langage de contraintes. Cette étude est aussi en rapport avec le travail antérieur de François Pottier sur l'inférence de types avec sous-typage et le travail de thèse de Daniel Bonniot sur la formalisation de $ML_{\leq}$.

6.1.7 Typage des enregistrements

Participant : Didier Rémy.

Didier Rémy étudie deux nouvelles formes de types enregistrements, les enregistrements *multi-dimensionnels* et les enregistrements à *étiquettes structurées*. Ces travaux sont récents et prospectifs.

Les enregistrements *multi-dimensionnels* sont des tableaux multi-dimensionnels à champs nommés hétérogènes. À la différence des enregistrements d'enregistrements, qui fixent un ordre de projection, les enregistrements multi-dimensionnels 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 la réception de ce message. L'avantage de cette approche est de considérer cette table à la fois comme un ensemble de classes définissant chacune un ensemble de méthodes ou comme un ensemble de méthodes chacune définie sur un ensemble de classes: on peut alors ajouter une nouvelle méthode (à un ensemble de classes) ou une nouvelle classe (avec un ensemble de méthodes). Cette solution permettrait de typer les fonctions génériques de Scheme et de corriger la dissymétrie créée par l'approche "objets comme enregistrements de méthodes".

Une autre idée consiste à ne plus considérer les étiquettes comme des atomes mais comme des entités structurées. La structure d'arbre est un cas particulier intéressant dont une application ambitieuse serait de représenter la hiérarchie des classes dans les étiquettes pour définir des méthodes simultanément pour une classe et ses sous-classes.

6.1.8 Typage de programmes incomplets

Participants : Michel Mauny, Walter Williams [New York University].

Michel Mauny et Walter Williams (étudiant à NYU) se sont intéressés aux programmes incomplets, et au problème de leur typage statique. L'idée qu'ils examinent est de partir de la technique de représentation d'analyses syntaxiques de programmes incomplets par des forêts partagées proposée par Bernard Lang en 1988, et de comprendre comment les utiliser pour effectuer le typage. L'automne a été essentiellement consacré à faire le tour du problème, et à comprendre les travaux sur ce domaine. Michel Mauny et Walter Williams continueront à collaborer sur ce sujet en 2001.

6.1.9 Types dépendants pour la programmation

Durant cette année 2000, Michel Mauny a exploré des voies de recherches ouvertes assez récemment qui consistent à utiliser des versions plus ou moins restreintes des types dépendants dans les langages de programmation (alors qu'ils n'ont été essentiellement utilisés jusqu'ici que dans des langages de spécification, ou des assistants à la preuve de programmes).

Jusqu'ici, les travaux les plus marquants dans ce domaine sont ceux de Zenger ou bien de Xi et Pfenning, ou encore ceux d'Augustsson. Zenger ou Xi et Pfenning utilisent des versions restreintes de types dépendants, assez puissantes pour exprimer des propriétés intéressantes, mais suffisamment restreintes pour autoriser une vérification statique décidable. En revanche, le langage Cayenne d'Augustsson utilise la pleine puissance des types dépendants dans un langage purement fonctionnel. Dans ce cas, le prix à payer est l'indécidabilité de la vérification du typage, en échange d'une très grande expressivité.

Michel Mauny explore actuellement l'utilisation de solutions différentes qui utiliseraient des vérifications dynamiques pour pallier l'indécidabilité de la vérification statique des types dépendants.

6.1.10 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 vérifications 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.

En 1999 la vérification statique des types d'emploi des fonctions génériques avait conduit à modifier l'algèbre de types du polymorphisme extensionnel, pour y intégrer les versions abstraites des fonctions génériques, 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 continué l'étude du difficile 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 régler le problème de cohérence. Dans les cas complètement ambigus, un message d'erreur est émis. Nous avons plus particulièrement étudié cette année une simplification drastique qui semble prometteuse: l'analyse statique unifierait systématiquement le type d'emploi de la fonction générique avec le premier filtre compatible (avec ce type sous-instancié). Cette méthode simple demanderait sans doute la réécriture de certaines fonctions génériques puisque l'ordre des filtres aurait ainsi une importance accrue. Cependant, cette modification est cohérente avec la sémantique habituelle du filtrage de ML et ne fait que renforcer l'intuition du programmeur ML qui écrit des fonctions génériques. En outre elle permet une compilation plus efficace du filtrage des arguments de types des fonctions génériques puisque les types passées à l'exécution sont supprimés, remplacés par des arbres de décision qui code directement les *aiguillage*

à suivre dans les filtrage des fonctions génériques. Ces arbres d'aiguillage ont une propriété sémantique forte, puisqu'ils sont isomorphes à la dérivation de typage qui a permis de prouver statiquement le bon typage d'un appel de fonction générique.

Principalement occupé par la rédaction de sa thèse, Jun Furuse a aussi étudié cette compilation efficace du filtrage des fonctions génériques à l'aide d'arbres d'aiguillage; il a montré que la nouvelle compilation était plus simple et plus efficace en terme de taille des données manipulées et de vitesse du code exécuté. June Furuse a écrit un article décrivant cette nouvelle méthode de compilation du filtrage des fonctions génériques [10]. Cet article sera présenté à la conférence JFLA'01.

L'introduction des valeurs dynamiques dans le polymorphisme extensionnel 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 constitue une voie de recherche intéressante.

6.1.11 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 et le typage des sommes polymorphes en Objective Caml. 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 Systèmes de modules

6.2.1 Modules et «mixins»

Participants : Xavier Leroy, Tom Hirschowitz.

Le langage Objective Caml est doté d'un système de modules évolué, reposant sur les notions de structures (collections de définitions nommées), signatures (collections de spécifications de types), et foncteurs (structures paramétrées par des structures). Néanmoins, ce système présente encore quelques limites. Ainsi, l'impossibilité de définir des valeurs mutuellement récursives dans des modules différents pose souvent des problèmes aux programmeurs. De

même, des mécanismes d'héritage et de redéfinition des composantes d'un module manquent au système actuel.

Lors de son stage de DEA, encadré par Xavier Leroy, Tom Hirschowitz a conçu un système primitif de modules *mixins* typés [17] qui apporte une solution au problème de la récursivité à travers les modules. Un prototype écrit en Objective Caml a été réalisé pour ce système.

Tom Hirschowitz poursuit ce travail dans le cadre d'une thèse de doctorat. Il a en particulier travaillé sur l'extension des modules mixins avec des notions d'héritage et de liaison tardive.

En parallèle avec ce travail sur les modules «mixins», qui pourrait déboucher sur la prochaine génération de modules pour Objective Caml, Xavier Leroy a complété l'étude du système de modules actuel d'Objective Caml, en publiant une implémentation de référence d'un typeur pour ce système de modules [4]. Cette implémentation est elle-même modulaire, en ceci qu'elle n'impose pas un langage de base particulier, mais est paramétrisée par une description et un typeur pour le langage de base considéré. Outre ses vertus pédagogiques, cette implémentation de référence démontre de manière entièrement constructive l'applicabilité de ce système de modules à une vaste classe de langages. Elle sert également de base pour le prototypage d'extensions du système de modules; en particulier, Tom Hirschowitz l'a utilisée pour son prototype de systèmes de modules «mixins».

6.3 Analyses statiques

6.3.1 Analyse de flots d'information

Participants : Sylvain Conchon [projet Moscova], François Pottier.

Supposons qu'un système informatique contienne des données confidentielles, pour des raisons militaires, commerciales ou personnelles. Comment en empêcher un mauvais usage? Il nous faut les protéger contre les programmes que nous utilisons, puisque ceux-ci pourraient contenir – intentionnellement ou non – des fragments de code qui permettraient la diffusion de certaines données confidentielles.

On a donc cherché, depuis plusieurs décennies, des moyens de contrôler la dissémination d'information effectuée par un programme. Une solution prometteuse consiste à analyser ces programmes, de façon à prouver, avant l'exécution, qu'ils se comporteront de façon acceptable. Ce procédé est généralement nommé analyse de *flots d'information*.

Les premiers pas dans cette direction datent de la fin des années 1970. Bien qu'intéressants, ces travaux ne proposent encore que des techniques d'analyse manuelles et dont la correction n'est pas prouvée. Il faut attendre le milieu des années 1990 pour voir cette analyse formulée en termes de *typage*.

François Pottier et Sylvain Conchon se sont attachés à la définition d'un système de types pour l'analyse de flots d'information dans un langage purement fonctionnel. Ils ont montré qu'un tel système peut être dérivé, de façon systématique, à partir d'un système de types standard, et ont fourni une preuve élémentaire de ce résultat. Ceci permet, moyennant un effort de preuve minime, d'appliquer à l'analyse de flots d'information toutes les techniques de typage classiques: polymorphisme, sous-typage, etc.

Un article décrivant ces travaux a été présenté à la conférence ICFP'00 [12].

6.3.2 Analyse de flots d'information pour un langage impératif

Participants : François Pottier, Vincent Simonet.

François Pottier encadre depuis septembre 2000 le stage de DEA de Vincent Simonet. L'objectif en est d'abord d'étendre les résultats théoriques obtenus par François Pottier et Sylvain Conchon [12] à un langage doté de références et d'exceptions, puis, dans la mesure du possible, d'écrire un analyseur de flots d'information pour l'ensemble du langage Caml-Light.

6.3.3 Détection statique de levée d'exceptions

Participants : Xavier Leroy, François Pessaux [Stevens Institute].

Ce travail porte sur la réalisation d'un analyseur statique de programmes écrits en Objective Caml, qui détecte les exceptions levées et non rattrapées interrompant inopinément l'exécution des 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 est plus fine que les autres analyses au niveau des arguments des exceptions et des structures de données, ce qui donne une meilleure précision globale que les analyses habituelles qui doivent restreindre leurs approximations pour conserver des temps d'analyse raisonnables. Ainsi, l'analyseur réalisé montre que l'utilisation de techniques «plus faibles» donne en pratique des résultats au moins aussi acceptables que ceux d'autres analyses basées sur des cadres théoriquement plus précis.

François Pessaux a soutenu sa thèse portant sur ce travail, le 16 décembre 1999 à l'université de Paris 6. Ce travail s'est achevé cette année par la mise à disposition publique d'un prototype d'analyseur statique des exceptions d'Objective Caml.

6.3.4 Contrôle d'accès statique pour Java

Participants : François Pottier, Christian Skalka [John Hopkins University, Baltimore], Scott Smith [John Hopkins University, Baltimore].

Le langage Java est doté d'un mécanisme de contrôle d'accès, pour garantir que certaines ressources sensibles (fichiers, périphériques, réseau, etc.) ne sont accessibles qu'aux acteurs (aussi appelés *principaux*). Ce mécanisme est implanté de façon purement dynamique: lorsque la machine virtuelle doit effectuer un contrôle, elle parcourt la pile d'appels pour vérifier que toutes les méthodes en cours d'exécution ont été signées par des principaux autorisés. Cette inspection de la pile est coûteuse, comme l'a signalé Wallach. De plus, les garanties de sécurité fournies par un fragment de programme Java sont peu claires, dans la mesure où les opérations d'autorisation et de contrôle d'accès sont dispersées au sein du code.

Christian Skalka et Scott Smith ont proposé, dans un article publié à ICFP'00, de pallier ces problèmes en effectuant le contrôle d'accès de façon statique, grâce à un système de types *ad hoc*. François Pottier a collaboré avec eux pour améliorer cette proposition initiale. Il a montré qu'un tel système de types pouvait être dérivé, de façon systématique, à partir de la traduction *security-passing style* déjà proposée par Wallach. Cette approche permet de construire un systèmes de types avancé (dont plusieurs variantes ont été étudiées), moyennant une quantité de preuves fort réduite.

Un article décrivant ces travaux a été soumis à la conférence ESOP'01.

6.3.5 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, université 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.

L'année dernière Pascal Cuoq a défini et 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 était 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 (utilisée dans Lustre) est d'*expanser* 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.

Cette année Pascal Cuoq a donc travaillé sur la génération de code pour les programmes Lucid Synchrone qu'accepte son analyse de causalité. Le polymorphisme et l'ordre supérieur du langage rendent l'ordonnancement statique complet du code généré impossible à obtenir, mais différentes méthodes sont envisagées pour réduire le surcoût dû à l'expressivité de l'analyse de causalité. En particulier, on envisage d'utiliser le partitionnement du code généré en blocs pour lesquels l'évaluation est stricte, une technique qui rappelle celle que Pascal Raymond décrit dans son rapport de DEA.

6.4 Implémentations de Caml

6.4.1 Objective Caml

Participants : Xavier Leroy, Damien Doligez [projet Moscova], Jacques Garrigue [université de Kyoto], Luc Maranget [projet Moscova], Jérôme Vouillon [université de Pennsylvanie], Pierre Weis.

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 publique la version 3.00 d'Objective Caml. La principale innovation de cette version est l'ajout au langage des arguments de fonctions nommés ou optionnels, très utiles pour auto-documenter et améliorer la sûreté d'utilisation de bibliothèques complexes. De surcroît, un nouveau mécanisme de "sommes ouvertes" permet maintenant la manipulation de valeurs de types sommes sans déclaration préalable du type, avec un haut degré de polymorphisme, et un typage par inférence de types très précis. Ces deux nouveaux traits, initialement développés par Jacques Garrigue (université de Kyoto) dans son prototype OLabl, ont été intégrés à OCaml par ce dernier lors de son séjour au projet Cristal fin 1999. De retour à l'université de Kyoto, il continue à en assurer le développement et la maintenance.

La version 3.00 d'Objective Caml ajoute également un mécanisme général de "finalisation" de structures Caml, dû à Damien Doligez (projet Moscova). Damien Doligez et Xavier Leroy ont aussi enrichi la bibliothèque standard, et amélioré les mécanismes d'interfaçage entre Caml et C: comparaison, sérialisation et désérialisation de structures de données implémentées en C.

Après la publication de la version 3.00, le développement d'Objective Caml s'est poursuivi par un travail important sur le compilateur. Luc Maranget a considérablement amélioré la compilation du filtrage (définitions par cas). Xavier Leroy a écrit un générateur de code pour la nouvelle architecture IA64 d'Intel. Ces travaux sont décrits respectivement dans le rapport d'activité du projet Moscova et dans la section «Objective Caml sur IA64» ci-dessous.

6.4.2 Objective Caml sur IA64

Participant : Xavier Leroy.

La société Intel conçoit depuis deux ans une nouvelle architecture de processeurs 64 bits, nommée IA64. Cette architecture introduit plusieurs traits originaux, notamment le contrôle explicite du parallélisme d'instructions (par opposition aux processeurs super-scalaires actuels, qui extraient eux-mêmes et de manière transparente le parallélisme d'un code purement séquentiel), la possibilité de rendre conditionnelles sur un prédicat des instructions arbitraires, et des mécanismes de préchargement de données explicites. Ces traits, destinés à faciliter la réalisation de processeurs de hautes performances, exigent davantage d'optimisations de la part du compilateur; les exploiter correctement représente un défi majeur pour les compilateurs.

Xavier Leroy a réalisé un portage d'Objective Caml sur l'architecture IA64, comprenant un nouveau générateur de code et un nouvel ordonnanceur d'instructions qui extrait le parallélisme présent au niveau des blocs de base du programme. Bien que le processeur Itanium (la première implémentation de l'architecture IA64) ne soit pas encore commercialement disponible, ce portage a pu être testé et mesuré sur des machines Itanium prototypes mises par Intel à la disposition des auteurs de logiciels libres via l'organisation SourceForge. Ceci fait d'Objective Caml l'un des tout premiers langages compilés de haut niveau qui tournent sur l'architecture IA64.

À la suite de ce travail, Xavier Leroy a commencé à réfléchir sur l'exploitation des autres traits de l'architecture IA64, notamment les instructions conditionnelles et le préchargement des données. Les langages fonctionnels comme Objective Caml, de par leur sémantique mieux contrôlée et leur plus faible nombre de modifications en place de structures de données, pourraient mieux se prêter à l'exploitation effective de ces traits architecturaux que les langages classiques comme C ou Java. Xavier Leroy a présenté ces travaux au groupe de travail IFIP 2.8 «Programmation fonctionnelle».

6.4.3 Interopérabilité

Participants : Xavier Leroy, Bruno Pagano [projet Moscova], Manuel Serrano [université de Nice], Luc Maranget [projet Moscova].

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.

L'interopérabilité entre Objective Caml et d'autres langages, soit directement, soit via une architecture standard de composants logiciels (Corba, COM, COM+) est donc l'une de nos priorités. Cette année, nous avons réalisé les développements suivants sur ce thème:

- Dans le cadre d'un contrat avec Microsoft Research, Bruno Pagano, sous la direction de Xavier Leroy et Luc Maranget, a réalisé un prototype d'intégration d'Objective Caml dans la nouvelle architecture COM+ de Microsoft.

  COM+ fournit un environnement d'exécution commun (machine virtuelle, compilateurs *just-in-time*, allocateur mémoire, glaneur de cellules, bibliothèques standard) qui peut servir de cible à de nombreux langages (C++, C#, Visual Basic, Java, Eiffel, Scheme, Haskell, Mercury, ...) et favorise ainsi leur interopérabilité avec partage des structures de données. COM+ est à la base de l'architecture «.NET» rendue publique par Microsoft en juillet 2000.

  Le prototype de Bruno Pagano compile Objective Caml vers le code de la machine virtuelle COM+. Il obtient des performances honorables, intermédiaires entre celles de

l'interprète de bytecode Caml et celles du compilateur natif Caml.

COM+ étant un formalisme typé à la Java, l'intégration d'Objective Caml pose des problèmes difficiles liés aux différences fondamentales entre le système de types de COM+ et celui d'Objective Caml (équivalence par nom vs. équivalence structurelle, en particulier). Le prototype de Bruno Pagano contourne ce problème via un codage supplémentaire de certaines structures de données, mais ce codage empêche d'atteindre l'objectif initial d'interopérabilité par partage des données, car il introduit des étapes supplémentaires de codage et de décodage des données lors des communications inter-langages.

- En collaboration avec le Commissariat à l'Énergie Atomique, direction des réacteurs nucléaires, Xavier Leroy et Manuel Serrano ont ajouté à Objective Caml des mécanismes facilitant l'interopérabilité avec des programmes de calcul numérique écrits en Fortran. Ces mécanismes sont décrits en détail en section 6.4.4. Ce travail joue un rôle important dans l'un des projets en cours au CEA: l'utilisation d'Objective Caml comme langage de contrôle et de couplage de grosses applications de calcul numérique (cf. section 6.4.4).

- Xavier Leroy a poursuivi le développement du générateur de code d'interface CamlIDL, dont la version 1.01 a été publiée en août 2000. 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++ conformément au modèle COM de Microsoft (le prédécesseur de COM+). CamlIDL permet 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).

- Enfin, Xavier Leroy a commencé le développement d'une interface 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.

  Tout comme l'intégration Caml/COM+, cette interface Caml/Java pose d'intéressants problèmes de correspondance entre les systèmes de types et de classes des deux langages, qui sont significativement différents. Le codage des types et des classes Java en termes de types et de classes Caml utilise des traits avancés de Caml, et constitue un excellent test «grandeur nature» de la partie à objets d'Objective Caml.

6.4.4 OCaml et tableaux Fortran

Participants : Xavier Leroy, Manuel Serrano [université de Nice], Damien Doligez [projet Moscova].

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, et Xavier Leroy ont travaillé sur une extension de OCaml qui comble ce manque. Ils ont élaboré et implémenté une interface de programmation pour l'utilisation de matrices Fortran et C en OCaml. 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.

6.4.5 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.

6.4.6 Transformations de programmes par règles de réécritures

Participant : Dick Kieburtz.

Dick Kieburtz, professeur à l'Oregon Graduate Institute, a rejoint le projet Cristal pour un séjour de 4 mois à partir du mois de mars 2000.

Il a étudié des méthodes compositionnelles pour implémenter des transformations de programmes par systèmes de réécriture automatique des termes. Après avoir implémenté une transformation de déforestation sur un petit langage fonctionnel (avec définitions récursives), Dick Kieburtz a découvert le besoin impératif de développer une logique pour raisonner sur les

propriétés des stratégies automatisées de réécritures. Cette réflexion a donné lieu à l'écriture d'un article intitulé *A Logic for Rewriting Strategies*.

6.4.7 Ajout de contraintes temps-réel dans la programmation des systèmes réactifs

Participants : Dick Kieburtz, Marc Pouzet [université Paris 6].

Dick Kieburtz a collaboré un jour par semaine avec Marc Pouzet (université Paris 6) pour mieux comprendre le rapport entre les systèmes réactifs synchrones (avec horloges) et les systèmes à contraintes temps réel sans horloges. Le résultat de ce travail prospectif mène actuellement à l'écriture d'un article sur "Les réactions sensibles au temps" (*Time-Sensitive Reactions*).

6.4.8 Documentation de l'implémentation de Caml.

Participants : Didier Le Botlan, Alan Schmitt [projet Moscova].

Alan Schmitt (projet Moscova) et Didier Le Botlan ont commencé à rédiger une documentation technique de l'actuelle implémentation de Caml. L'objectif est de fournir un document permettant de comprendre assez rapidement les quelques milliers de lignes de code composant le système Caml.

Cette documentation est disponible à l'adresse suivante : `http://cristal.inria.fr/~lebotlan/docaml.html`.

6.5 Applications de Caml

6.5.1 Implémentation d'un langage de scripts en Caml

Participant : Bruno Verlyck.

Depuis le mois de décembre, Bruno Verlyck collabore une journée par semaine avec le projet Cristal, pour développer une bibliothèque qui permettrait l'écriture de scripts système en Caml. Cette bibliothèque, nommée *Cash* pour **Ca**ml **sh**, 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`). On envisage d'utiliser Camlp4 pour ajouter une surcouche syntaxique qui facilite encore l'écriture des scripts. Pour l'instant le travail consiste à réaliser les couches basses de la bibliothèque (fonctions d'interfaçage système de haut niveau, définies à l'aide des fonctionnalités du module `Unix`, l'interface du langage Caml avec le système).

6.5.2 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 est distribuée dans le recueil des bibliothèques d'Objective Caml, le *bazar OCaml*.

6.5.3 Affichage de DAG en HTML

Participant : Daniel de Rauglaudre.

Daniel de Rauglaudre a écrit le logiciel `dag2html` qui permet d'afficher des DAG (*Directed Acyclic Graphs*) simples en HTML. Le code produit utilise une table HTML pour représenter le DAG, des balises <HR> pour les lignes horizontales et des caractères "barre verticale" pour les lignes verticales, l'ensemble étant "élastique" horizontalement, comme le sont les tables HTML.

Ce programme est utilisé dans le logiciel GeneWeb pour l'affichage de DAG généalogiques.

6.5.4 Serveur Web de bases de données généalogiques

Participant : Daniel de Rauglaudre.

Daniel de Rauglaudre a continué le développement de GeneWeb, logiciel multilingue de gestion de bases de données généalogiques entièrement écrit en OCaml, qui fonctionne avec une interface Web, soit en serveur Web soit en CGI.

De très nombreux traits ont été ajoutés, principalement dans le but de rendre le logiciel plus simple, plus complet, plus rapide ou plus agréable à utiliser. Certains de ces traits sont dûs à la collaboration avec les utilisateurs du logiciel, en particulier avec deux programmeurs, Ludovic Ledieu et Louis Granboulan. Daniel de Rauglaudre a installé son programme `dag2html` (voir ci-dessus) dans GeneWeb, ce qui permet de voir des arbres généalogiques divers: liens de parenté par ancêtre commun (avec éventuellement plusieurs branches combinées), par alliance (chemin le plus court entre deux personnes), affichage des liens entre les personnes ayant un même titre. Daniel de Rauglaudre a aussi ajouté plusieurs systèmes pour personnaliser les pages. En particulier un système de fichiers-modèles dans l'esprit des feuilles de style de HTML, mais avec une syntaxe différente (moins verbeuse et mieux adaptée) et un principe différent puisque les pages HTML sont produites par le serveur et non pas interprétées par le navigateur du client. Des modifications ont été apportées dans le logiciel pour permettre son utilisation sur CD-ROM, sans nécessiter la copie du logiciel ou de la base généalogique sur le disque dur. Quatre langues ont été ajoutées: le finlandais, le russe, l'afrikaans et le letton, ce qui porte le nombre de langues disponibles à 19. La documentation a été corrigée et complétée sur quelques points, avec des contributions d'utilisateurs: Lars Gustavsson et Yann Corno.

Daniel de Rauglaudre a écrit en OCaml le logiciel `gwtp` (*GeneWeb Transfer Program*) qui permet aux clients d'un site GeneWeb d'envoyer ou de télécharger leurs bases de données en utilisant l'interface de leur navigateur, ce qui évite l'intervention directe de l'administrateur du site.

Daniel de Rauglaudre gère aussi le serveur `geneweb.inria.fr`, le site pilote de Geneweb, qui permet de tester le logiciel en vraie grandeur sur plus de cinquante bases de données généalogiques.

GeneWeb est un logiciel libre et gratuit dont les sources sont distribuées sous la licence GNU. Les distributions comportent une version Windows, une version Linux, un paquetage RPM RedHat, RPM LinuxPPC et Debian (la version Debian est écrite par un utilisateur: Christian Perrier), une version Solaris, et les fichiers de code source. Onze nouvelles versions ont été distribuées au cours de l'année 2000. Toutes les informations sur GeneWeb sont à l'adresse `http://pauillac.inria.fr/~ddr/GeneWeb/`.

6.5.5 Concours ICFP

Participants : Sébastien Ailleret [projet Moscova], Pascal Cuoq, Damien Doligez [projet Moscova], Robert Harley, Fabrice Le Fessant [projet Moscova], Alan Schmitt [projet Moscova].

Un programme écrit en Objective Caml par Sébastien Ailleret, Pascal Cuoq, Damien Doligez, Robert Harley, Fabrice Le Fessant, Xavier Leroy et Alan Schmitt a obtenu le second prix au concours de programmation organisé à l'occasion du congrès ICFP 2000.

Le premier prix a été attribué à un programme également écrit en Objective Caml par une équipe de l'université de Pennsylvanie dirigée par Jérôme Vouillon, ancien doctorant du projet Cristal et contributeur important à l'implémentation d'Objective Caml.

6.6 Compilation de sémantiques à réduction

Participant : Michel Mauny.

Yong Xiao, étudiant de l'*University of Oregon* à Eugene (Oregon, USA), avait conçu un langage de spécification de sémantiques à réduction ainsi que leur compilation lors d'un séjour qu'il a effectué chez Cristal durant l'été 1998. Ce travail avait été réalisé en collaboration avec Xavier Leroy, Didier Rémy et Michel Mauny.

À l'occasion de son séjour aux USA, Michel Mauny a continué ce travail avec Yong Xiao et Zena Ariola (UOregon), et notamment participé à la rédaction des résultats dans un article présenté au groupe de travail RULE 2000 [15].

6.7 Preuves de code mobile

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

6.7.1 Preuves de code mobile, bytecode typé, compilation de Coq

Co-encadré par Xavier Leroy et Benjamin Werner (projet Coq), Benjamin Grégoire s'intéresse à la technologie du code auto-certifié de P. Lee et G. Necula (*Proof-Carrying Code* ou *PCC*) et au langage assembleur typé de G. Morisett, K. Crary, D. Walker et N. Glew (*Typed Assembly Language* ou *TAL*). Il a cherché à comprendre quel rapport il y a entre ces deux techniques de certification de programmes compilés.

À la suite de ces réflexions, on envisage de compiler le langage Coq vers un bytecode typé. Cette technologie déboucherait sur une nouvelle méthode de preuve de programmes, à cheval

entre PPC et TAL, en héritant directement de tous les outils de preuves de programmes dont dispose déjà Coq.

6.8 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.

Ces résultats ont rendu possible en 1999 le calcul d'un logarithme discret sur une courbe elliptique définie sur $\mathrm{GF}(2^{97})$, le plus difficile jamais réalisé à l'époque.

Cette année Robert Harley s'est principalement consacré à la rédaction de sa thèse.

6.8.1 ECDL

Participants : Robert Harley, Daniel de Rauglaudre, Damien Doligez [projet Moscova], Xavier Leroy.

Cette année, Robert Harley s'est attaqué à ECC2K-108, un défi cryptographique reposant sur le calcul d'un logarithme discret sur une courbe elliptique. Ce défi, lancé par la société Certicom, fait suite au calcul réalisé en 1999.

Daniel de Rauglaudre a participé activement au projet en écrivant un programme CGI en OCaml qui a permis aux nombreux contributeurs de l'Internet de voir en temps réel l'évolution du projet ainsi que la contribution de leurs ordinateurs.

La base de données des résultats était mise à jour tous les quarts d'heure par des programmes écrits en shell et en OCaml par Daniel de Rauglaudre. Damien Doligez et Xavier Leroy ont aussi participé à cet effort sans précédent en réalisant des outils systèmes pour la gestion des résultats et en aidant à la réalisation d'un code de calcul extrêmement efficace.

Obtenu avec l'assistance de milliers d'internautes, ce résultat a fait l'objet de plusieurs communiqués de presse et a été relaté dans de nombreux médias. Il contribue de manière signifiacive à l'étalonnage des algorithmes de chiffrement à clés publiques reposant sur des courbes elliptiques.

6.9 Linguistique computationnelle

Participant : Gérard Huet.

6.9.1 État de l'art de la linguistique computationnelle

Gérard Huet s'est fixé comme objectif pour 2000 de comprendre l'état de l'art du domaine *Linguistique computationnelle* aussi appelé *Traitement automatique des langues naturelles.*

Cette année a donc été mise à profit pour 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. Un certain investissement dans le domaine proprement linguistique étant nécessaire, une partie de l'activité a été d'apprendre les bases de cette discipline, notamment par la lecture d'ouvrages de de Saussure, Wittgenstein, Tesnière, Lyons, Austin, Bourdieu, Chomsky, Mel'čuk, Pustejovsky, Pierrel et Pinker. Durant le mois d'Août, G. Huet a également assisté, en tant qu'auditeur, à l'Ecole ESSLLI'2000 organisée à Birmingham par l'European Association for Logic, Language and Information, ce qui lui a permis de rencontrer un certain nombre d'experts du domaine. Il a également assisté en mai au workshop TAG+5, ce qui lui a permis de rencontrer les principaux acteurs de la technologie "Grammaires d'Arbres Adjoints".

Un programme de visites de certains laboratoires d'excellence du domaine a été entrepris, étant bien entendu qu'une approche exhaustive était hors de question. Ont ainsi été visités:

- en janvier à Boston, le laboratoire LCS du Pr Dertouzos au MIT; le Pr Stuart Shieber au Maxwell Dworkin Lab et le Pr Michael Witwel au Dept de Sanskrit d'Harvard University, Candy Sidner à Lotus Development Labs, et Sam Christy, CEO de Wordstream Inc.
- en février discussions avec Laurent Romary, chef du projet Langue et Dialogue du LORIA, et avec Patrice Lopez du DFKI de Sarrebruck.
- en mars à Pune (Maharashtra) visite du Pr V. N. Jha, directeur du CASS (Center for the Advanced Study of Sanskrit), du Dr Hemant Darbari du CDAC (Center for Development of Advanced Computing) et de Anand Deshpande, Managing Director de Persistent Systems Ltd.
- en mai discussions à Paris avec Anne Abeillé de l'université Paris 7, Mark Steedman de l'université d'Edinburgh, et Aravind Joshi de l'University of Pennsylvania.
- fin mai visite du Pr Aarne Ranta à l'université de Göteborg.
- en juin à l'occasion du Natural Language Tutorial. de la conférence CADE à Pittsburgh, discussions avec Michael Kolhase.
- en août à Birmingham (ESSLLI), discussions avec Pr Stephen Pulman de l'université de Cambridge, Christian Rétoré (INRIA-Rennes), Bob Krowetz des laboratoires NEC de Princeton, Pr Ruth Kempson de King's College à Londres, Dick Crouch de Xerox-PARC, Pr Mary Wood de l'université de Manchester, et Pr Johan van Benthem de l'université d'Amsterdam.
- en septembre discussions avec Maurice Gross du LADL (Paris 7) et avec Philippe Lefevre du laboratoire de recherches d'EDF à Clamart.

- en octobre visite du laboratoire XRCE de Xerox à Grenoble, discussions avec Pierre Isabelle, Lauri Karttunen, Kenneth Beesley et Marc Dymetman

- fin octobre visite de l'université de Sarrebruck, du projet VerbMobil au DFKI, et du Max Planck Institute; discussions avec Pr Manfred Pinkal, Pr Wolfgang Wahlster, Pr Jörg Siekmann, Dr Dieter Hutter, Dr Erica Melis, Dr Kristoph Benzmuller, Armin Fiedler; présentation détaillée du produit VerbMobil par Reinhard Karger.

- en novembre visite du laboratoire CLIPS de l'IMAG à Grenoble, discussions avec Jean Caelen et Christian Boitet; visite de Célestin Sedogbo au LCR Thomson-CSF; participation à la journée "Langues et Technologies" du RNRT organisée à Nancy par Jean-Marie Pierrel.

6.9.2 Structure du Lexique

Gérard Huet a étudié le moyen concret de structurer une base de données lexicale pour le sanscrit. Cette expérience a donné lieu à la mise en place d'un site Web qui a été mis en service en février; il autorise la consultation interactive d'un dictionnaire français-sanscrit à l'aide d'un indexage par technologie CGI-OCaml. Voir `http://pauillac.inria.fr/~huet/SKT/DICO`. Ce travail a été présenté début avril à la XIème conférence internationale de sanscrit à Turin. Un rapport de recherche INRIA [18] décrit la structure de la base de données.

Gérard Huet a aussi mené une étude approfondie des déclinaisons des substantifs du sanscrit, et a programmé un processeur de sandhi (allitération phonétique). Ce programme permet la consultation grammaticale en ligne des mots fléchis, à l'aide d'une interface CGI-OCaml accessible depuis le dictionnaire, ou indépendamment depuis la page `http://pauillac.inria.fr/~huet/SKT/decls.html`. Ce travail a mené à la conception d'une plate-forme modulaire de traitement de la langue centrée autour d'une base de données lexicale structurée. L'architecture de cette plate-forme est discutée dans un article en préparation [3].

La conclusion de ce travail, pertinente pour le projet Cristal, est que l'ensemble d'outils OCaml et Camlp4 est bien adapté au développement d'une plate-forme modulaire fiable et efficace pour le traitement des langues naturelles.

7 Contrats industriels (nationaux, européens et internationaux)

7.1 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.

7.2 Microsoft: interopérabilité

Ce contrat, avec la société Microsoft, a porté sur l'interopérabilité entre langages dans le cadre de la plate-forme .NET, et sur l'intégration d'OCaml dans celle-ci. Il a financé le travail de Bruno Pagano, et Xavier Leroy en a été 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 *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.

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

Dick Kieburtz, professeur à l'Oregon Graduate Institute (Portland, Oregon), est venu pour 4 mois de mars à juin 2000.

Vassily Litvinov, doctorant à l'université de Washington, est venu faire un stage au sein du projet Cristal d'octobre à fin décembre 2000.

Michel Mauny a passé une année sabbatique aux États-Unis, à l'université NYU (New York University), où il était *Visiting Professor* dans le département d'Informatique, au Courant Institute.

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 des jurys de recrutement CR2 des unités de Rocquencourt et de Rennes.

Michel Mauny est membre élu (suppléant) de la Commission d'Évaluation. À ce titre, il a participé au jury du concours DR2 au printemps 2000.

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

Gérard Huet est responsable pour l'INRIA des relations scientifiques avec l'Inde. À ce titre il s'est rendu en Inde en mars et en décembre, et il organise un programme d'échanges d'étudiants avec les IITs (stages d'été, mise au point d'un programme de master en cotutelle entre le MTech de l'IIT de Delhi et le DEA Programmation géré par l'université Paris 7).

Michel Mauny a été «contact chercheurs» de l'UR de Rocquencourt pour la campagne de recrutement du début 2001. Son rôle était de renseigner et d'orienter les candidats potentiels vers les équipes de l'INRIA susceptibles d'être intéressées par leur candidature.

9.1.2 Animation de la communauté scientifique hors INRIA

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

Au mois de décembre, Gérard Huet a organisé en Inde un workshop SCILAB avec l'équipe Metalau à l'Institut de Physique de Bhubaneswar (Orissa).

Didier Rémy a été membre du comité de direction de la conférence ICFP (*International Conference on Functional Programming*) jusqu'en juin 2000, date d'expiration de son mandat de trois ans.

9.1.3 Animation de la communauté des usagers de Caml

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 Marseille au colloque de l'association des professeurs de mathématiques des classes préparatoires.

9.1.4 Consortium Caml

Le Consortium Caml a pour vocation de réunir les industriels et académiques désireux d'aider et de pérenniser le langages et les développements autour du langage Caml.

Michel Mauny a terminé l'élaboration des textes du Consortium avec l'aide de Jean-Louis Bouchenez et des services administratif et juridique de l'INRIA. Les textes seront rendus publics au début l'année 2001 à `http://caml.inria.fr/consortium/`.

9.1.5 Comités de lecture et programmes

Xavier Leroy a fait partie du comité de programme des congrès Programming Language Design and Implementation 2000, International Conference on Functional Programming 2000, et Compiler Construction 2001.

Michel Mauny est membre du comité de lecture de Techniques et Sciences Informatiques, et a été membre du comité de programme du *Final Workshop* du groupe de travail APPSEM, qui doit avoir lieu en mars 2001 à Darmstadt.

Xavier Leroy présidera le comité de programme du congrès International Conference on Functional Programming pour l'année 2001.

9.1.6 Jurys de thèses

Michel Mauny a été membre du jury de d'habilitation à diriger les recherches de Pascal Fradet (IRISA), ainsi que du jury de thèse de doctorat de Jérôme Vouillon (Paris 7).

Xavier Leroy a fait partie du jury d'habilitation de Manuel Serrano (université de Nice, septembre 2000) et du jury de thèse de Bruno Blanchet (École Polytechnique, décembre 2000).

Didier Rémy a fait partie du jury de thèse de Jérôme Vouillon (Paris 7).

9.1.7 Autres activités d'intérêt général

Diffusion des connaissances

Participant : Pierre Weis.

Pierre Weis a réalisé une quatrième et cinquième 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 3000 exemplaires cette année.

9.2 Enseignement

9.2.1 Encadrement et jurys

Didier Rémy a encadré la thèse de Jérôme Vouillon qui a soutenu au mois d'octobre; depuis le mois de septembre il encadre la thèse de Didier Le Botlan. Didier Rémy a aussi encadré le stage de Vassily Litvinov en visite du 5 octobre au 25 décembre.

Pierre Weis encadre la thèse de Jun Furuse.

Xavier Leroy et Benjamin Werner (projet Coq) co-encadrent la thèse de Benjamin Grégoire. Xavier Leroy a encadré le stage de DEA de Tom Hirschowitz, et encadre maintenant sa thèse.

Michel Mauny encadre les thèses de Robert Harley et de Pascal Cuoq (ce dernier est co-encadré par Marc Pouzet, LIP6).

François Pottier encadre depuis septembre 2000 le stage de DEA de Vincent Simonet.

9.2.2 Enseignements de troisième cycle universitaire

Xavier Leroy assure le cours Typage et Programmation (20h) dans le DEA «Programmation: Sémantique, preuves et langages» (Paris 6-7-11, X, ENS)).

Michel Mauny est membre de l'équipe de direction de ce DEA, et Xavier Leroy le remplace temporairement à ce poste pour l'année 2000–2001.

François Pottier a assuré deux séances (4h) du cours *Typage et sous-typage* de Giuseppe Castagna, dans le cadre du DEA Programmation: Sémantique, Preuves et Langages. Il a également effectué des travaux dirigés à l'École Nationale Supérieure des Techniques Avancées (ENSTA); il s'agissait d'une initiation à Java pour des élèves de première année (14h).

Michel Mauny a donné 3 cours de niveau *graduate* à New York University (*Programming Languages*, *Foundations of Modern Type Systems*, et *Honors Programming Languages*), pour un volume horaire total d'une centaine d'heures de cours magistral.

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 sur la modularité et un cours de compilation dans le cursus de deuxième année.

Didier Rémy 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.

Didier Le Botlan participe à l'encadrement des travaux dirigés du Tronc Commun d'informatique des élèves de l'École Polytechnique pour un total de 44 heures, ainsi que des séances d'initiation qui représentent 20 heures.

Benjamin Grégoire a assuré des TP de programmation à l'École Polytechnique pour un total de 44 heures.

9.2.4 Enseignements de premier et second cycles universitaires

Tom Hirschowitz est moniteur en IUP 1 à l'université Paris 7 pour le cours d'architecture des microprocesseurs (64h).

Daniel Bonniot effectue un monitorat à l'université Paris 7, et enseigne en TD/TP en premier cycle (64h).

Benjamin Grégoire a assuré des TD et TP de langage C et d'architecture, en DEUG, à l'université Paris 7.

9.3 Participation à des colloques, séminaires, invitations

Xavier Leroy et Didier Rémy ont participé au congrès Principles of Programming Languages (Boston, janvier 2000).

Didier Rémy a participé aussi au workshop *Foundations of Object Oriented Languages (FOOL)* de janvier à Boston. Il a donné une conférence invitée *Re-exploring multiple inheritance* [19] au workshop FOOL.

Xavier Leroy a participé au groupe de travail sur COM+ organisé par Microsoft Research à Cambridge en mars 2000, et y a présenté l'avancement des travaux sur l'interface Caml/COM+.

François Pottier a assisté à la conférence ESOP'00 à Berlin en avril 2000, et y a présenté certains résultats concernant l'inférence de types avec contraintes [13]. Une version remaniée et étendue de cet article est à paraître dans le *Nordic Journal of Computing* [7].

François Pottier et Xavier Leroy ont participé à la réunion de juillet 2000 du groupe de travail IFIP 2.8 «Programmation fonctionnelle» (Packwood, Washington). Xavier Leroy y a présenté son travail sur la génération de code pour l'architecture IA64, et François Pottier ses travaux sur l'analyse de flot d'informations.

Xavier Leroy et François Pottier ont participé au congrès International Conference on Functional Programming (Montréal, septembre 2000). François Pottier y a présenté ses travaux avec Sylvain Conchon [12].

Daniel Bonniot, Pascal Cuoq, Tom Hirschowitz, Didier Le Botlan et Vincent Simonet ont assisté à l'école d'été *International summer school on applied semantics* organisée par l'INRIA et l'université de Minho qui s'est tenue du 9 au 15 septembre 2000 à Caminha au Portugal,

dans le cadre du groupe de travail européen APPSEM. Didier Rémy y a donné un cours *Using, Understanding, and Unravelling the Ocaml language* [20].

Didier Rémy a participé au séminaire CONFER qui a eu lieu à Stockholm en Juin.

Pascal Cuoq a assisté à l'école d'été "Méthodes Formelles" CEA-EDF-INRIA en juillet 2000.

Dick Kieburtz, Pascal Cuoq, June Furuse et Pierre Weis ont assisté aux Journées Francophones des Langages Applicatifs, en février 2000 au Mont Saint-Michel. Jun Furuse y a fait la présentation de son article avec Pierre Weis sur les entrées-sorties de valeurs sûres pour ML [9]. Dick Kieburtz a donné une conférence invitée, intitulée «Coalgebraic Techniques for Reactive Functional Programming» [11].

Dick Kieburtz a fait un bref séjour à l'ENS-Lyon et à l'université de Franche-Comté (Besancon). Il s'est aussi rendu en visite à l'université d'Utrecht (Hollande) et au laboratoire Microsoft Research de Cambridge (Angleterre).

9.4 Relations industrielles

Xavier Leroy est le responsable (*principal investigator*) du contrat avec Microsoft sur l'interopérabilité.

Xavier Leroy et François Pottier sont consultants auprès de la société Trusted Logic, respectivant pour 1 jour par semaine et un jour par mois.

Pierre Weis est consultant un jour par semaine auprès de la société LexiFi Technologies.

10 Bibliographie

Livres et monographies

[1] E. Chailloux, P. Manoury, B. Pagano, *Développement d'applications avec Objective Caml*, O'Reilly, 2000.

Thèses et habilitations à diriger des recherches

[2] J. Vouillon, *Conception et réalisation d'une extension du langage ML avec des objets*, thèse de doctorat, Université Paris 7, octobre 2000.

Articles et chapitres de livre

[3] G. Huet, *Computational Linguistics for Sanskrit: a Software Engineering Approach*, décembre 2000, à paraître dans le volume en l'honneur de Rod Burstall.

[4] X. Leroy, « A modular module system », *Journal of Functional Programming 10*, 3, 2000, p. 269–303, `http://pauillac.inria.fr/~~xleroy/publi/modular-modules-jfp.ps.gz`.

[5] F. Pessaux, X. Leroy, « Type-based analysis of uncaught exceptions », *ACM Transactions on Programming Languages and Systems 22*, 2, 2000, p. 340–377, `http://pauillac.inria.fr/~~xleroy/publi/exceptions-toplas.ps.gz`.

[6] F. Pottier, «Simplifying subtyping constraints: a theory», *Information & Computation*, août 2000, accepté pour publication, à paraître, `http://pauillac.inria.fr/~~fpottier/publis/fpottier-ic-2000.ps.gz`.

[7] F. Pottier, «A Versatile Constraint-Based Type Inference System», *Nordic Journal of Computing*, novembre 2000, accepté pour publication, à paraître, `http://pauillac.inria.fr/~~fpottier/publis/fpottier-njc-2000.ps.gz`.

Communications à des congrès, colloques, etc.

[8] 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/`.

[9] J. Furuse, P. Weis, «Entrées-sorties de valeurs en Caml», *in : Actes des Journées francophones des langages applicatifs*, INRIA, janvier 2000.

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

[11] R. Kieburtz, «Coalgebraic Techniques for Reactive Functional Programming», *in : Actes des Journées francophones des langages applicatifs*, INRIA, janvier 2000. Conférence invitée.

[12] F. Pottier, S. Conchon, «Information Flow Inference for Free», *in : Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP'00)*, p. 46–57, Montréal, Canada, septembre 2000, `http://pauillac.inria.fr/~~fpottier/publis/fpottier-conchon-icfp00.ps.gz`.

[13] F. Pottier, «A 3-part type inference engine», *in : Proceedings of the 2000 European Symposium on Programming (ESOP'00)*, G. Smolka (éditeur), *Lecture Notes in Computer Science, 1782*, Springer Verlag, p. 320–335, mars 2000, `http://pauillac.inria.fr/~~fpottier/publis/fpottier-esop-2000.ps.gz`.

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

[15] Y. Xiao, Z. Ariola, M. Mauny, «From Syntactic Theories to Interpreters: A Specification Language and Its Compilation», *in : First International Workshop on Rule-Based Programming (RULE 2000)*, N. Derschowitz, C. Kirchner (éditeurs), sep 2000, `http://xxx.lanl.gov/abs/cs.PL/0009030/`.

Rapports de recherche et publications internes

[16] X. Leroy, D. Rémy, J. Vouillon, J. Garrigue, D. Doligez, *The Objective Caml system, documentation and user's manual – release 3.00*, INRIA, avril 2000, documentation distribuée avec le système Objective Caml, `http://caml.inria.fr/ocaml/htmlman/`.

Divers

[17] T. Hirschowitz, *Modules mixins : typage et implantation*, Mémoire de DEA, Université Paris 7, 2000.

[18] G. Huet, «Structure of a sanskrit dictionary», novembre 2000, rapport de Recherche INRIA, à paraître.

[19] D. Rémy, «Re-exploring multiple inheritance», Conférence invitée à FOOL'7, janvier 2000.

[20] D. Rémy, «Using, Understanding, and Unravelling the Ocaml Language. From theory to Practice and vice versa.», Notes de cours pour l'école d'été APPSEM'2000, Caminha, Portugal, septembre 2000, `http://cristal.inria.fr/~~remy/cours/appsem/`.

[21] D. Rémy, J. Vouillon, «On the (un)reality of virtual types», à paraître en rapport de recherche INRIA, 2000.

[22] P. Weis, «Logiciels libres distribués par l'INRIA», INRIA, janvier 2000, CD-ROM.

[23] P. Weis, «Logiciels libres distribués par l'INRIA», INRIA, avril 2000, CD-ROM.