INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet Cristal

Programmation typée, modularité et compilation

Rocquencourt

THÈME 2A

Table des matières

1. Composition de l'équipe

Responsable scientifique
Michel Mauny [DR, INRIA]

Responsable permanent
Pierre Weis [DR, INRIA]

Assistante de projet (commune avec Logical)
Nelly Maloisel
Stéphanie Aubin [à partir d'octobre 2002]

Personnel INRIA
Gérard Huet [DR]
Xavier Leroy [DR]
François Pottier [CR]
Daniel de Rauglaudre [IR]
Didier Rémy [DR]
Bruno Verlyck [IR, temps partiel]

Chercheur associé
Roberto Di Cosmo [Prof. Paris 7 en détachement]

Ingénieur associé
Maxence Guesdon [ODL jusqu'en août 2002, IR Miriad à 20% chez Cristal depuis le 1/09/2002]

Collaborateur extérieur
Robert Harley [Inscrit en doctorat à l'Université Paris 7]

Doctorants
Daniel Bonniot [AMN, université Paris 7]
Pascal Cuoq [Boursier INRIA, université Paris 6, jusqu'en octobre 2002]
Jun Furuse [Boursier du gouvernement japonais, université Paris 7]
Benjamin Grégoire [Allocataire MENRT, université Paris 7, commun avec Logical]
Tom Hirschowitz [Allocataire MENRT, université Paris 7]
Didier Le Botlan [AMX, université Paris 7]
Vincent Simonet [Normalien, université Paris 7]

Stagiaires
Dimitri Ara [université Paris 6, du 1/07 au 15/08/2002]
Gaurav Chanda [IIT Dehli, du 2/05 au 31/06/2002]
Avik Chaudhuri [IIT Dehli, du 2/05 au 31/06/2002]
Romain Théret [UVSQ, du 19/06 au 15/09/2002]
Boris Yakobowski [ENS Lyon, du 4/06 au 31/07/2002]

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).

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

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

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

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

3.1.1. Synthèse de types de ML

Le langage ML dispose, depuis sa conception en 1978, d'un système de types qui permet la synthèse automatique des types à la compilation, et qui dispose aussi d'une forme de polymorphisme qui autorise la programmation d'algorithmes génériques (c'est-à-dire d'algorithmes qui acceptent de traiter toute une variété de données de types différents). Le système de types de ML est spécifié formellement, et un théorème énonce sa propriété principale : aucune erreur de type ne peut se produire durant l'exécution d'un programme bien typé.

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

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

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

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

3.1.2. Types d'ordre supérieur

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

3.1.3. Typage des objets

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

L'une des approches que suit le projet Cristal repose sur les travaux de Didier Rémy, qui a proposé dans sa thèse un système d'enregistrements extensibles qui se prête bien à la synthèse de types. Après la conception et la réalisation d'une première maquette d'un système à objets pour ML appelé ML-ART[33], 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 que représente la synthèse des types. La puissance de ce système de types permet ainsi de typer des programmes qui sont réputés difficiles à typer, même dans les langages où l'information de types est explicite.

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

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

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

Enfin la dernière approche que nous explorons repose sur la surcharge d'identificateurs et les fonctions génériques, ces fonctions dont le comportement est guidé par le type avec lequel elles sont utilisées. C'est la théorie du polymorphisme extensionnel, sujet de thèse de Jun Furuse. Cette approche permet aussi de

formaliser les calculs qui dépendent dynamiquement des types comme les entrées-sorties ou les valeurs dynamiques.

3.1.4. Typage et confidentialité

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

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

Là encore, l'utilisation de systèmes de types est féconde. François Pottier et Sylvain Conchon ont tout d'abord défini un système de types pour l'analyse de flots d'information dans un langage purement fonctionnel, ce qui permet d'appliquer à l'analyse de flots d'information toutes les techniques de typage classiques : polymorphisme, sous-typage. François Pottier et Vincent Simonet ont ensuite poursuivi ce travail en l'étendant à un langage de programmation plus réaliste doté de traits impératifs tels que références et exceptions, Core ML, puis en réalisant un prototype, Flow Caml, qui permet une analyse automatique de programmes écrit en Caml. L'utilisation de cette première implémentation les amène maintenant à chercher à étendre l'expressivité du système afin d'obtenir un outil plus flexible.

3.2. Le langage Caml

Glossaire

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

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

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

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

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

Compilation et exécution Le compilateur d'un langage de programmation traduit les programmes écrits dans ce langage en des instructions de plus bas niveau, interprétables par la machine. On distingue les instructions de *code par octets* (ou *bytecode*) et les instructions de *code natif*. Les instructions de bytecode sont indépendantes de la machine qui exécute le programme : elles définissent une véritable **machine virtuelle** qui est réalisée par un programme dédié qui exécute les instructions de bytecode, l'*interprète de*

bytecode. Le même jeu d'instructions de bytecode est donc utilisable pour une grande variété d'architectures de machines : toutes les architectures pour lesquelles on dispose de l'interprète de bytecode. En revanche, les instructions de **code natif** sont spécifiques à l'architecture matérielle de la machine d'exécution : le jeu d'instructions est celui de la machine d'exécution.
Ces deux types de code présentent des avantages et des inconvénients différents : s'il produit du bytecode, le compilateur du langage de haut niveau est simplifié puisqu'il ne doit produire qu'un seul type d'instructions pour tous les types de machines. S'il produit du code natif, le travail du compilateur est bien plus difficile puisque le code produit change pour chaque type de machine, mais en revanche le code obtenu est plus rapide à l'exécution puisque les instructions du code natif sont directement interprétées par la machine.
Le **modèle d'exécution** du langage est une vision abstraite de l'exécution du code produit par le compilateur et de sa politique de gestion de la mémoire.

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[34] joue un rôle important. L'équipe en tire en effet des sujets de recherche et l'utilise non seulement comme champ d'expérimentation et de validation, mais aussi comme moyen privilégié de transfert et de diffusion des résultats.

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

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

Les recherches menées au projet Cristal ont conduit à l'adjonction au langage Caml de traits importants comme les modules et les objets. En effet, ces deux façons de structurer les programmes ont été l'objet de recherches intensives afin de les doter de sémantiques statiques (typage) et dynamiques (exécution) clairement définies et prouvées correctes. Les objets ont été étudiés par Didier Rémy (cf. section 3.1.3). Les modules ont été étudiés par Xavier Leroy[31], 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[32]. 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[29]. Lors du passage de Caml-Light à Objective Caml, Xavier Leroy a ajouté au générateur de bytecode un tout nouveau compilateur qui produit du code natif performant. L'alliance de ces deux compilateurs permet dorénavant de disposer du meilleur des deux mondes : portabilité et cycle de développement réduit grâce au bytecode, performance et efficacité grâce au code natif que produit le compilateur optimisant (qui est disponible pour les architectures les plus courantes).

4. Domaines d'application

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.

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.

On constate cependant 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.

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

5. Logiciels

5.1. Implantations de Caml

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

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

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

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

5.2. Outils syntaxiques

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

6. Résultats nouveaux

6.1. Systèmes de types

6.1.1. Extension de ML avec des types de second ordre

Participants : Didier Rémy, Didier Le Botlan.

Le langage ML manipule des types de premier ordre (sans quantificateurs), enrichis par des schémas de types qui autorisent la quantification universelle, mais seulement de façon externe. Cela permet de synthétiser les types très simplement par un calcul d'unification de premier ordre, mais limite leur expressivité.

Symétriquement, le système F autorise des types de second ordre (les quantificateurs peuvent être imbriqués à une profondeur arbitraire) ce qui lui confère une plus grande expressivité, mais rend la synthèse des types indécidable.

Didier Le Bottlan et Didier Rémy ont étudié un système mixte ML^F qui combine la synthèse des types de ML et l'expressivité du système F. Ce travail généralise une proposition antérieure de Didier Rémy et Jacques Garrigue [30], qui permettait déjà d'encapsuler des types de second ordre dans un système de premier ordre, mais en maintenant une différence entre les schémas de types inférés et les types explicites de second ordre indiqués par l'utilisateur.

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

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

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

6.1.2. *Types polymorphes contraints*

Participants : Didier Rémy, Daniel Bonniot.

Dans le cadre de sa thèse *Extension et mise en oeuvre des systèmes de types polymorphes contraints* sous la direction de Didier Rémy, Daniel Bonniot a travaillé à améliorer les systèmes de types pour langages à multi-méthodes. Il a publié ses travaux antérieurs sur une présentation stratifiée de ces systèmes de types dans [11] et a rédigé une version journal de cet article qui a été soumise à publication. Cette version a donné l'occasion de développer la formalisation des modules et de la compilation séparée. Daniel Bonniot a finalisé l'ajout de contraintes de *kinding* qu'il avait commencé l'année dernière ; cela a donné lieu à la publication d'un article [12] ainsi qu'à la soumission d'une version journal.

Daniel Bonniot a commencé le traitement de l'opérateur *super* dans le cadre des multi-méthodes. Dans les langages à mono-méthodes, *super* permet d'appeler une méthode de la classe parente même si cette méthode a été redéfinie dans la classe courante. Puisqu'une multi-méthode, contrairement à une mono-méthode, n'est pas attachée à une classe privilégiée, cette sémantique ne peut être reprise telle quelle. Les langages Dylan et Cecil ont la notion d'appel de l'implémentation suivante dans la liste des implémentations classées par spécificité décroissante. Cette sémantique permet de redéfinir une méthode dans un cas précis en réutilisant l'implémentation plus générale. Daniel Bonniot a commencé une formalisation de cette sémantique dans le cadre de [11]. Elle permettra de montrer la sûreté de cette construction dans un riche cadre formel.

Daniel Bonniot a implémenté ces différentes extensions théoriques dans le langage Nice (http://nice.sourceforge.net) afin d'en vérifier la faisabilité et l'utilité pratique.

6.1.3. *Synthèse de types avec rangées à base de contraintes*

Participant : François Pottier.

François Pottier a réalisé une analyse de complexité pour l'algorithme de résolution de contraintes de sous-typage en présence de rangées qu'il a proposé voici deux ans, ce qui clôt une question restée ouverte. De plus, ce travail a permis d'imaginer une nouvelle présentation de l'algorithme, plus simple, et propice à une nouvelle compréhension. Ce travail a fait l'objet d'un article soumis à la conférence LICS'03.

6.1.4. *Inférence de types en présence de sous-typage structurel*

Participants : François Pottier, Vincent Simonet.

Le sous-typage structurel est une forme particulière de sous-typage dans laquelle deux types comparables doivent partager le même squelette ou *structure*. Cette forme de sous-typage apparaît naturellement lorsqu'un système de types basé sur l'unification est étendu en annotant les types par des *atomes* appartenant à un treillis, dans le but, par exemple, d'effectuer une analyse statique telle qu'une analyse de flots de données ou d'information, ou bien d'exceptions non rattrapées.

La conception d'un synthétiseur de types pour un système doté de sous-typage et de polymorphisme peut généralement être décomposée en deux parties relativement distinctes. La première étape est spécifique au système étudié mais reste souvent simple à implémenter. Elle consiste à parcourir l'arbre syntaxique du programme analysé et à générer incrémentalement une contrainte. La deuxième partie nécessite de résoudre et de simplifier les contraintes, pour des raisons d'efficacité et de lisibilité. Vincent Simonet s'est intéressé à la mise en œuvre d'algorithmes efficaces pour résoudre ces problèmes dans le cas du sous-typage structurel. Pour cela, il a adapté et combiné un certain nombres de techniques existantes, dont celles développées par François Pottier dans sa thèse pour une autre forme de sous-typage. La correction du système obtenu a été prouvée. Ce travail a été décrit dans un article soumis à la conférence *ESOP'03* [28].

Vincent Simonet a parallèlement implémenté ces algorithmes en Objective Caml et ainsi réalisé une bibliothèque générique pour l'inférence de types en présence de sous-typage structurel (http://cristal.inria.fr/~simonet/soft/dalton/). Cette librairie se présente sous la forme d'un *foncteur* paramétré par une série de modules décrivant l'algèbre de types du client. Ainsi, nous espérons qu'elle pourra être utilisée comme moteur d'inférence dans une variété d'applications. Cette bibliothèque a en particulier été utilisée pour la réalisation de *Flow Caml*.

6.1.5. Isomorphismes de types avec variants

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

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

Un exemple simple de tels isomorphismes est très connu dans la programmation fonctionnelle sous le nom d'isomorphisme de Curry : $A \longrightarrow (B \longrightarrow C) \cong (A \times B) \longrightarrow C$.

Cette étude a des applications pratiques immédiates dans la conception d'outils de recherche de fonctions dans de librairies de programmation (ou dans la recherche de lemmes dans des bibliothèques d'assistants à la programmation), qui ont été effectivement incorporés dans des versions récentes de Caml-Light et de Coq en collaboration avec David Delahaye, Xavier Leroy, Jérome Vouillon, Pierre Weis et Benjamin Werner

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

En coopération avec Marcelo Fiore, de l'Université de Cambridge (UK), Vincent Balat et Roberto Di Cosmo ont étudié les isomorphismes de types en présence de sommes (types variants) : il s'agit d'un problème bien plus complexe des précédents et qui était ouvert depuis longtemps, mais ils ont pu obtenir pour la première fois une série de résultats significatifs dans ce domaine.

Ils ont montré qu'une famille d'égalités entières connues sous le nom d'identités de Gurevich sont en effet des isomorphismes : cela permet d'un côté de prouver qu'en présence des sommes fortes les isomorphismes ne sont pas finiment axiomatisables, et de l'autre de donner un contenu combinatoire à des identités dont la preuve jusque là n'était pas élémentaire. En même temps, ils ont montré qu'il existe des identités entières faisant intervenir les sommes et le type vide qui ne sont pas des isomorphismes.

Un article rendant compte de ces recherches a été présenté à la conférence LICS 2002, et est repris en détail dans la thèse de Vincent Balat, qui a été soutenue le 5 décembre 2002.

6.1.6. Isomorphismes de signatures dans OCaml

Participants : Roberto Di Cosmo, Boris Yakobowski.

Boris Yakobowski, stagiaire de l'ENS-Lyon encadré par Roberto Di Cosmo, a travaillé cet été sur l'implémentation d'un prototype de moteur de recherche dans des librairies de modules OCaml basé sur la notion d'ismorphisme de signatures proposée par María-Virginia Aponte, Catherine Dubois et Roberto Di Cosmo. Ce travail a produit tout d'abord une implémentation de la procédure de filtrage par isomorphismes qui peut aisément être intégrée dans des outils de programmation OCaml existants. Ensuite, Boris Yakobowski a pu compléter un prototype qui nous sert de base pour expérimenter diverses solutions et divers algorithmes pour la génération des foncteurs OCaml de conversion entre signatures isomorphes.

Ce travail a été présenté au *workshop* international WIT'02.

6.1.7. Isomorphismes de types en présence de sous-typage et types récursifs

Participants : Roberto Di Cosmo, François Pottier, Didier Rémy.

Roberto Di Cosmo, François Pottier et Didier Rémy se sont intéressés aux isomorphismes de types produits (*e.g.* produits cartésiens, enregistrements, *etc.*), en présence de types récursifs et de sous-typage. La prise en compte du sous-typage est particulièrement intéressante dans le cas des langages à objets où une fonction qui possède un type possède également tous ses sous-types. La prise en compte des seuls isomorphismes entre produits semble également suffisante pour couvrir les cas les plus pertinents en pratique.

Un algorithme pour la comparaison de types dans ce contexte a été conçu et prouvé correct. Un prototype a également été réalisé pour vérifier que l'algorithme est suffisamment efficace en pratique. La correction de cet algorithme et l'analyse de sa complexité sont en cours d'étude.

6.1.8. Présentation de l'inférence de types pour ML étendu

Participants : François Pottier, Didier Rémy.

François Pottier et Didier Rémy rédigent actuellement une présentation exhaustive du typage et de l'inférence de types pour ML et ses extensions. Celle-ci doit faire partie d'un livre (à paraître) dont la rédaction est supervisée par Benjamin Pierce.

6.1.9. Polymorphisme extensionnel et fonctions génériques

Participants : Pierre Weis, Jun Furuse.

Jun Furuse a consacré l'année 2002 à la rédaction de sa thèse. Celle-ci a été soutenue en décembre 2002.

6.2. Enregistrements, modules et mixins

6.2.1. Modules et mixins

Participants : Xavier Leroy, Tom Hirschowitz, Joe Wells [Heriot-Watt University].

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

Un langage de programmation se doit de fournir des outils pour les deux processus. Le langage OCaml possède un puissant système de modules, permettant la compilation séparée. Cependant, sur les deux plans évoqués (décomposition intuitive des programmes et réutilisation du code), des progrès restent à faire.

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

Dans les deux cas, des astuces existent pour contourner les problèmes, au prix de la sûreté d'exécution et de l'élégance.

Dans ce contexte, des systèmes primitifs de modules *mixins* ont été formulés, qui permettent les dépendances mutuelles entre modules, l'héritage et la redéfinition. Toutefois, ils se placent dans le contexte de langages en appel par nom, où toutes les formes de récursion sont autorisées. Ce n'est pas le cas en OCaml, où seules les valeurs peuvent être récursives (typiquement, les fonctions). Tom Hirschowitz et Xavier Leroy ont mis au point un système de typage pour les modules mixins appelé CMS_v, qui permet d'éviter les écueils de la récursion mal fondée, tout en conservant l'expressivité souhaitée. Ils ont de plus mis en place un schéma de compilation pour ce système, compatible avec la compilation séparée, vers le langage λ_B, un λ-calcul avec enregistrements et `let rec`. Néanmoins, ce schéma de compilation génère des termes habituellement rejetés par tous les systèmes de typage du λ-calcul. Tom Hirschowitz et Xavier Leroy ont donc mis au point un nouveau système de typage pour λ_B, qui accepte les programmes générés par le schéma de compilation, tout en garantissant leur sûreté. Les résultats de cette recherche sont un article présenté au congrès ESOP 2002 [14] et sa version longue, avec les preuves de sûreté [25], soumise à publication.

Ce travail se poursuit en collaboration avec Joe Wells par la mise au point d'une sémantique à réduction sûre pour les modules mixins (par opposition à la sémantique par traduction vers λ_B), ainsi qu'une étude de la compilation du `let rec`. Ce travail est en cours d'élaboration.

6.2.2. Enregistrements et modules

Participant : François Pottier.

Les langages de la famille ML disposent d'un langage *de modules*, qui forme une couche distincte au-dessus du langage *de base*. Un module est un ensemble de définitions de valeurs et de types. Le langage de modules permet également de définir des fonctions des modules dans les modules (foncteurs), augmentant ainsi la ré-utilisabilité du code.

Malgré de nombreux travaux, les langages de modules aujourd'hui disponibles restent complexes et d'une flexibilité limitée. François Pottier souhaiterait donc augmenter la puissance du langage de modules, d'une part en le ré-unifiant avec le langage de base, ce qui devrait diminuer la complexité de l'ensemble, d'autre part en le munissant d'un système de types plus expressif.

Pour cela, il a défini un nouveau langage noyau, lequel est une extension conservative de ML, d'une part, et (au prix de quelques annotations) contient le langage F_ω, d'autre part. Un prototype de moteur d'inférence de types a été réalisé pour ce langage. Ces résultats nouveaux sont en cours de rédaction.

6.3. Analyses statiques

6.3.1. Analyse de flots d'information

Participants : François Pottier, Vincent Simonet.

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

François Pottier et Vincent Simonet ont poursuivi leurs travaux relatifs à l'analyse de flots d'information sous plusieurs formes.

L'an passé, Vincent Simonet et François Pottier ont réalisé une analyse de flots d'information pour un langage fonctionnel doté de traits impératifs (i.e. de références) et d'exceptions, comparable au langage Caml-Light, et établi sa correction. Ce travail a tout d'abord été exposé à la conférence *Principles of Programming Languages* (POPL) en janvier 2002 [20]. Une présentation étendue et améliorée a ensuite été acceptée pour publication par le journal *ACM Transactions on Programming Languages and Systems* [8].

Vincent Simonet s'est également intéressé à une extension de cette analyse offrant un traitement particulièrement fin des types sommes, et par traduction, des mécanismes d'exceptions. Outre l'originalité du système de types présenté, l'intérêt de cette étude réside dans le fait qu'elle a permis de mieux comprendre et de comparer les systèmes précédents dont le traitement des exceptions avait été développé de manière relativement *ad hoc*. Cependant, sa complexité intrinsèque laisse peu d'espoir quant à la possibilité d'une implantation complète réaliste. Ce travail [21] a été présenté au *IEEE Computer Security Foundations Workshop* qui s'est tenu au Canada en juin 2002.

Parallèlement à ces travaux théoriques, Vincent Simonet a implémenté un analyseur de flots d'information pour le langage Caml, *Flow Caml*, en se basant sur le système développé dans [20]. Dans cette extension, les types habituels de Caml sont annotés par des « niveaux ». Les flots d'information sont alors décrits par des contraintes de sous-typage entre ceux-ci. L'inférence de type est préservée, aussi bien sur la structure des types que sur les annotations, ce qui permet de ne pas annoter les programmes et de disposer d'une analyse automatique. La quasi-totalité des constructions du langage Caml sont traitées (à l'exception des objets, des labels et des variants polymorphes), ainsi que le système de modules. Une première version de cet analyseur est d'ores et déjà disponible (http://cristal.inria.fr/~simonet/soft/flowcaml/).

6.3.2. *Flots synchrones et programmation fonctionnelle*

Participants : Pascal Cuoq, Michel Mauny.

Pascal Cuoq, doctorant au sein du projet Cristal jusqu'au 31 octobre 2002, a rédigé et soutenu sa thèse intitulée « Ajout de synchronisme dans les langages fonctionnels fortement typés ». Les deux contributions de cette thèse sont une analyse de causalité pour un langage de flots synchrones d'ordre supérieur, et une méthode d'implémentation d'un interpréteur de flots synchrones sous la forme d'une extension du langage OCaml.

A partir du 1er novembre 2002, Pascal Cuoq a commencé un post-doctorat au sein du projet ROPAS, dans l'institut de recherche KAIST (Corée du Sud), dans le cadre du programme INRIA de bourses post-doctorales à l'étranger. Les deux thèmes de recherche du projet ROPAS sont la compilation des langages de la famille ML et les applications pratiques de l'interprétation abstraite. Pascal Cuoq a en particulier étudié les possibilités d'interfaçage entre les techniques d'interprétation abstraite et celles du proof-carrying code initiées par Lee et Necula.

6.4. Compilation certifiée de la réduction forte

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

Co-encadré par Xavier Leroy et Benjamin Werner (projet Logical), Benjamin Grégoire effectue ses travaux de thèse sur la compilation efficace des mécanismes de réduction forte et de test d'inter-convertibilité qui jouent un rôle important dans le système Coq. Un des traits fondamentaux du Calcul des Constructions (la logique sous-jacente du système Coq) est qu'il combine raisonnements et calculs dans un même formalisme. Ainsi, la validité d'une proposition logique peut dépendre du fait qu'un terme calculatoire qu'elle contient s'évalue bien en le résultat attendu. La vérification d'une preuve Coq nécessite donc l'exécution de calculs dans un langage fonctionnel pur. Dans certains types de preuves, le temps d'exécution de ces calculs est important et devient un facteur limitant pour la vérification de grosses preuves.

Le travail de thèse de Benjamin Grégoire consiste à accélérer ces calculs en les effectuant non plus par un inteprète (comme dans le système Coq actuel), mais via la compilation des termes vers le code d'une machine abstraite. Il s'agit là d'une approche bien connue dans le domaine de l'implémentation des langages fonctionnels comme Objective Caml. Cependant, le transfert de la technologie de compilation de Caml à ce problème se heurte à une difficulté de taille : alors que l'évaluation des langages fonctionnels effectue seulement de la réduction « faible » (on ne calcule pas dans les corps de fonctions non appliquées), les calculs effectués par le prouveur Coq nécessitent toute la puissance de la réduction « forte », où les réductions sont également effectuées à l'intérieur des fonctions.

En collaboration avec Xavier Leroy, Benjamin Grégoire a apporté une solution originale à ce problème : on décompose la réduction forte en une alternance de réduction faible symbolique (réduction faible en présence

de variables libres) et de relecture des valeurs produites ; de plus, la réduction faible symbolique s'effectue efficacement via compilation vers une machine abstraite dérivée de celle d'Objective Caml.

Benjamin Grégoire a implémenté entièrement cette approche dans le système Coq. Elle débouche sur des gains de vitesse considérables (facteur 40 environ) sur la vérification de preuves ayant un contenu calculatoire important, comme la preuve Coq du théorème des quatre couleurs de G. Gonthier et B. Werner. Ce travail a donné lieu à publication au congrès *International Conference on Functional Programming* 2002.

En parallèle, Benjamin Grégoire a également prouvé en Coq la correction de cette approche : compilation, machine abstraite, et relecture. Il est maintenant en train de rédiger sa thèse.

6.5. Traitement de données XML

6.5.1. OCamlDuce

Participants : Roberto Di Cosmo, Michel Mauny, Jérôme Vouillon [université Paris 7].

La langage XDuce (http://xduce.sourceforge.net/) est un langage fonctionnel du premier ordre spécialement conçu pour programmer des transformations statiquement typées de documents XML. XDuce est doté d'un puissant mécanisme de filtrage. Les types de XDuce sont très proches des DTDs et le typage de XDuce est explicite et utilise du sous-typage. XDuce a été initialement développé à UPenn, dans l'équipe de Benjamin Pierce, avec notamment Haruo Hosoya et Jérôme Vouillon (en postdoc dans cette équipe, à l'époque).

Afin de faire de XDuce un vrai langage de programmation, celui-ci a besoin de mécanismes d'exécution efficaces et de bibliothèques : autant de choses dont dispose OCaml. À l'inverse, OCaml ne peut ni implémenter ni typer des transformations de documents XML de façon aussi précise que XDuce, en raison des différences fondamentales entre leurs disciplines de typage.

L'objectif de cette action est donc d'associer OCaml et XDuce dans un même environnement de développement : OCaml apportant son efficacité et son caractère généraliste, XDuce apportant son caractère dédié aux traitements de documents XML. Roberto Di Cosmo, Michel Mauny et Jérôme Vouillon ont commencé à travailler sur ce sujet. Les premiers points abordés ont été la coexistence des deux modèles d'exécutions (comment voir une fonction XDuce comme une fonction OCaml) et comment ajouter simplement de l'ordre supérieur à XDuce (enrichir son algèbre de types) tout en préservant une sémantique simple au langage.

6.5.2. Édition distante de documents XML

Participants : Gérard Huet, Avik Chaudhuri.

Gérard Huet a encadré un stage d'été d'Avik Chaudhuri, étudiant de l'IIT de Delhi. Le sujet était la conception et l'implémentation d'un serveur générique de sessions Web utilisant la technologie CGI. Le serveur est implémenté comme programme CGI d'interface avec un démon UNIX gérant une base de données de services, clients, sessions. Les différentes requêtes d'une session sont mémorisées par un serveur spécifique à un service, le démon ne gérant que l'administration et l'aiguillage des requêtes. Ceci permet d'étendre de manière très simple le mécanisme d'appel de procédures distants du protocole HTTP en un véritable gestionnaire de processus distribués, sans nécessiter l'importation dynamique d'exécutables ni dans le client (applets Java) ni dans le serveur (Javascript).

Le serveur prototype a été entièrement écrit en Objective Caml. Il a été testé sur un éditeur interactif structuré de conception originale, utilisant la technologie des *zippers*. Cette validation constitue une preuve de faisabilité de ce qui pourrait évoluer en un véritable éditeur XML distribué, utilisable depuis un client navigateur HTML sans installation particulière, et sans exécution locale.

6.5.3. OCaml-SOAP

Participants : Gaurav Chanda, Michel Mauny.

Michel Mauny a encadré un stage d'été de Gaurav Chanda, étudiant de l'IIT de Delhi. Le sujet était de permettre d'interfacer des programmes OCaml avec le standard SOAP. Gaurav Chanda a commencé par étendre une bibliothèque existante permettant l'interfaçage avec XML-RPC, standard concurrent de SOAP, et sensiblement

plus simple. Il s'est ensuite attaqué à SOAP, et a produit une bibliothèque dotée des fonctionnalités minimales. SOAP est en effet trop complexe pour qu'il soit possible de réaliser une interface complète dans le temps imparti. Le travail de Gaurav Chanda est documenté et distribué à http://caml.inria.fr/ocaml-soap/. Maxence Guesdon a fourni une aide précieuse en ce qui concerne les scripts de configuration et d'installation.

6.5.4. IO-XML

Participant : Daniel de Rauglaudre.

Daniel de Rauglaudre a écrit IoXML, une extension de syntaxe de Camlp4 permettant de générer automatiquement des imprimeurs et des parseurs en format XML. Chaque déclaration de type génère alors deux fonctions, l'une pour l'impression et l'autre pour l'analyse syntaxique de données XML.

6.6. Implémentations de Caml

6.6.1. Objective Caml

Participants : Xavier Leroy, Jacques Garrigue [université de Kyoto], Damien Doligez [projet Moscova], Luc Maranget [projet Moscova], Jérôme Vouillon [CNRS, équipe PPS, université Paris 7], 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 publiques les versions 3.04, 3.05 et 3.06 d'Objective Caml. La principale innovation de ces versions est l'ajout de types polymorphes de première classe pour les méthodes et les champs d'enregistrements : méthodes et champs peuvent être déclarés avec des types contenant des variables de types explicitement quantifiées universellement ; le typeur d'Objective Caml infère automatiquement la bonne instance de ces types lorsqu'on appelle la méthode ou accède au champ. Il s'agit là d'une première implémentation des mécanismes d'inférence partielle pour types polymorphes de second ordre développés par Jacques Garrigue et Didier Rémy.

Les autres nouveautés des dernières versions d'Objective Caml incluent :

- L'ajout d'un mécanisme d'encapsulation hiérarchique de bibliothèques, permettant de regrouper plusieurs modules $M_1, ..., M_n$ compilés séparément en un seul module M ayant $M_1, ..., M_n$ comme sous-modules. Ce mécanisme permet de réduire la « pollution » de l'espace de noms des unités de compilation, et par ce biais facilite la gestion de bibliothèques de grande taille.
- Une implémentation plus efficace des données évaluées paresseusement. Ces valeurs sont maintenant traitées de manière spéciale par le glaneur de cellules (GC), qui est capable de compresser les chaînes d'indirections introduites par l'évaluation de ces données « paresseuses ». Ce mécanisme a été utilisé par Andrew Tolmach (Portland State University) dans une expérience d'utilisation du *back-end* d'Objective Caml pour compiler le langage Haskell.
- Extensions et améliorations de la bibliothèque standard, dont en particulier deux nouveaux modules `Complex` (nombres complexes) et `Scanf` (entrées formattées). Ce dernier offre au programmeur Caml les fonctions de lecture structurée habituellement disponibles en langage C. La conception de cette bibliothèque est originale pour deux raisons : d'une part, l'interface en est entièrement fonctionnelle, sans affectations de variables ; d'autre part, les chaînes de format sont statiquement typées, en réutilisant de manière créative le typage existant des sorties formattées (module `Printf`).

6.6.2. Le préprocesseur Camlp4

Participant : Daniel de Rauglaudre.

Camlp4 est un préprocesseur pour OCaml, permettant de programmer des extensions syntaxiques dans OCaml, de changer la syntaxe du langage, de faire du formattage (*pretty print*) de programmes OCaml, et offrant un système de grammaires récursives descendantes dynamiquement extensibles.

Daniel de Rauglaudre a apporté cette année un certain nombre d'amélioration au système de grammaires extensibles de Camp4 : généralisation du type des lexèmes, ajout de méta-symboles grammaticaux permettant de programmer par exemple des itérateurs d'entrées grammaticales, conversion des imprimeurs de Camlp4 pour qu'ils utilisent les outils de formattage standard, traitement des commentaires à l'intérieur des phrases pour les différentes syntaxes concrètes d'OCaml.

Il a aussi amélioré ou ajouté différents kits d'analyse (qui dotent OCaml de syntaxes concrètes alternatives à la syntaxe standard) : une syntaxe à la Scheme pour les nostalgiques de parenthèses, amélioration de la syntaxe SML (qui permet de traduire des programmes SML en OCaml, testé sur le logiciel Tempo développé par le projet Compose (UR Futurs)).

Enfin, Daniel de Rauglaudre, en relation avec les utilisateurs de Camlp4, y a effectué de nombreuses autres petites améliorations, changements et corrections de bugs.

6.6.3. Autres bibliothèques

Participants : Jun Furuse, Pierre Weis.

En collaboration avec Pierre Weis, Jun Furuse a procédé à l'intégration des bibliothèques Caml/Tk et Labl/Tk, produisant ainsi une bibliothèque unifiée qui permet d'utiliser *simultanément* Caml/Tk et Labl/Tk, donc les deux styles de programmation. Ce travail, distribué avec le compilateur Objective Caml, a été réalisé avec l'aide matérielle de la société LexiFi.

En collaboration avec Pierre Weis, Jun Furuse a aussi continué à maintenir et tenir à jour les bibliothèques Camlimages, Caml/Tk et MMM. Ces bibliothèques sont distribuées dans le recueil des bibliothèques d'Objective Caml, le « bazar O'Caml ».

6.6.4. Outils de développement

6.6.4.1. Aide à la documentation

Participant : Maxence Guesdon.

OCamldoc, développé par Maxence Guesdon, est un outil permettant de générer de la documentation à partir de code source OCaml commenté. Cette année, OCamldoc a été intégré à la distribution du langage OCaml.

6.6.4.2. Environnement de développement OCaml

Participants : Maxence Guesdon, Dimitri Ara.

Maxence Guesdon a continué la réalisation de Cameleon, un environnement de développement pour le langage OCaml. Cameleon regroupe maintenant les autres outils de développement écrits par Maxence Guesdon (OCamlCVS, DBForge, Zoggy, Report, Configwin), qui étaient auparavant distribués séparément. De plus, Cameleon s'est enrichi des outils suivants :

Options une bibliothèque permettant de gérer des fichiers de configuration dans les applications développées en OCaml (avec Fabrice Le Fessant).

Okey une bibliothèque pour facilement ajouter des raccourcis clavier dans les applications utilisant la bibliothèque LablGtk.

Epeire une interface graphique pour le débogueur OCaml, permettant notamment la visualisation sous forme arborescente des structures manipulées par le programme débogué (avec Dimitri Ara).

Topcameleon une interface graphique pour le toplevel OCaml, avec la même possibilité d'affichage des valeurs que dans Epeire.

OCamlmake-o-matic un outil graphique pour construire des Makefiles (avec Dimitri Ara).

6.6.4.3. Outils et bibliothèque de génération de pages web

Participants : Maxence Guesdon, Pierre Weis.

Maxence Guesdon a développé `toolpage`, une collection d'outils et une bibliothèque permettant de générer des pages web. Cela comprend :

Toolpage un outil pour décrire des logiciels diffusés sur le web et qui génère la page principale de chaque outil, avec des liens vers la documentation, les archives de distribution, les pages associés, etc. Un exemple de page généré peut être vu à http://caml.inria.fr/devtools/.

Faquin un outil pour décrire des Foires Aux Questions sous forme arborescente et qui génère les pages web correspondantes. L'outil permet les références croisées entre questions (donc les liens internes à la FAQ sont vérifiés), ainsi que la possibilité de développer et d'utiliser (par chargement dynamique) d'autres générateurs (même technique que pour OCamldoc).

Toolhtml une bibliothèque pour générer des documents HTML, qui fournit des facilités pour construire des listes dans des tables, des arbres, des *frames*, ...

Egg un outil, encore en développement, pour décrire des structures de données et qui génère le code OCaml permettant de lire et écrire dans un fichier ces structures, ainsi que de les éditer graphiquement. Cet outil peut servir pour développer de petites applications qui fonctionnent sur le principe « édition des données - traitement ». Il ne reste à l'utilisateur qu'à écrire la partie « traitement », Egg génère le reste, directement compilable.

6.6.4.4. Htmlc

Cette année, Pierre Weis a distribué la bibliothèque *Htmlc* : c'est un outil de génération de pages web destiné à faciliter la maintenance et l'obtention d'un aspect uniforme aux pages d'un site. Maxence Guesdon l'utilise pour le site web qu'il développe pour les chercheurs de l'INRIA. À cette occasion, Maxence a commencé à participer au développement de l'outil.

6.7. Applications de Caml

6.7.1. Calcul parallèle

Participants : Roberto Di Cosmo, Xavier Leroy.

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

6.7.2. Implémentation d'un langage de scripts en Caml

Participant : Bruno Verlyck.

Bruno Verlyck a continué sa collaboration avec le projet Cristal, à raison de 4 journées par semaine, pour développer une bibliothèque permettant l'écriture de scripts en Caml. Cette bibliothèque, nommée **Cash** pour **Ca**ml **sh**ell, est inspirée du travail d'Olin Shivers sur Scsh (*a Unix Scheme shell*) (voir par exemple http://www.scsh.net/docu.html).

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

La première distribution publique a eu lieu en juin 2002, la seconde fin août. L'extension du module OCaml `Unix` est achevée.

6.7.3. Logiciel Active Dvi

Participants : Pierre Weis, Jun Furuse, Didier Rémy, Roberto Di Cosmo.

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

Active Dvi est documenté et librement distribué à partir de http://pauillac.inria.fr/advi/.

Pour Active DVI, l'année 2002 a été essentiellement consacrée à la maintenance du logiciel, et à l'ajout de caractéristiques en améliorant l'installation, la sécurité et la convivialité.

6.7.4. Serveur Web de bases de données généalogiques

Participant : Daniel de Rauglaudre.

GeneWeb est un logiciel de généalogie écrit en OCaml et qui fonctionne comme un service Web. Cette année, Daniel de Rauglaudre a effectué de nombreuses améliorations et corrections de bugs dans le logiciel GeneWeb. Une autre langue a été ajoutée, apportée par un contributeur (Mitja Iskric) : le slovène, et l'italien a été complété, en particulier en ce qui concerne la documentation du logiciel (Latino Imparato).

Le système de traductions du serveur de commandes interactives (`gwsetup`), a été simplifié, n'utilisant qu'un seul fichier-modèle par page d'affichage contenant toutes les traductions, au lieu de la multitude de fichiers pour chaque langue, ce qui rend la distribution plus petite (250 fichiers au lieu de 500), leur mise à jour plus facile, plus cohérente, et l'ajout de nouvelles langues plus facile. Du coup, ce serveur a été complété et peut traiter davantage d'options.

Daniel de Rauglaudre a distribué 3 nouvelles versions de GeneWeb, de 4.05 à 4.07. GeneWeb est disponible depuis http://cristal.inria.fr/~ddr/GeneWeb/.

6.7.5. Traduction en OCaml du logiciel Tempo

Participants : Daniel de Rauglaudre, Anne-Francoise Le Meur [projet Compose].

Daniel de Rauglaudre a converti le logiciel Tempo du projet Compose (UR Futurs) de SML en OCaml. Il a pour cela utilisé son logiciel Camlp4, en améliorant et complétant le kit d'analyse syntaxique de SML. Les 60000 lignes de SML se transforment en 65000 lignes d'OCaml avec conservation et réindentation des commentaires. La transformation a nécessité quelques modifications du programme SML initial et quelques ajustements du programme obtenu.

La version Tempo en OCaml fonctionne correctement. Des améliorations sont envisagées : suppressions des foncteurs inutiles qui alourdissent la compilation, transformation des objets en types enregistrements d'OCaml, simplification de l'évaluation gauche-droite des appels de fonctions (qui avaient dû être ajoutés) et certainement encore de nombreuses améliorations.

6.7.6. Couplage de codes numériques

Participants : Pierre Weis, Jérôme Jaffré [projet Estime], François Clément [projet Estime].

Avec les chercheurs du projet Estime, Pierre Weis participe à un groupe de travail sur l'élaboration d'un logiciel général destiné au « couplage de code ». Il s'agit d'utiliser de gros codes numériques travaillant sur des maillages de même dimension sur des parties disjointes d'un même maillage. Dans un premier temps on se limite au couplage de deux codes seulement. De façon générale, la valeur des champs sur la frontière entre les deux domaines interagit sur les valeurs des champs qui bordent la frontière, tandis que les valeurs des champs en dehors de la frontière permettent évidemment de calculer la valeur sur la frontière. Le couplage est considéré comme achevé lorsque la valeur du champ sur la frontière calculée depuis l'un ou l'autre des domaines bordés est la même (aux erreurs numériques près).

Le groupe de travail se réunit régulièrement, et le travail actuel consiste à définir précisément les structures de données manipulées par le coupleur et les formats d'échanges entre les programmes.

6.8. Arithmétique

Participants : Robert Harley, Michel Mauny.

Robert Harley a quitté l'INRIA à la fin 2000, mais a néanmoins continué à travailler sur son sujet de thèse dont le titre est *Advanced Algorithms for Arithmetic on Curves*. Il en a terminé la rédaction, et a soutenu le 16 décembre 2002.

6.9. Linguistique computationnelle

Participant : Gérard Huet.

Gérard Huet a abstrait de son travail d'analyse du sanskrit une boîte à outils implémentant les structures de base pour représenter un lexique, faire des traitements morphologiques ou phonologiques, et plus généralement implémenter des traitements d'état fini par des transducteurs rationnels guidés par le lexique. Ces traitements, entièrement réalisés dans un noyau applicatif d'Objectif Caml (appelé « Pidgin ML »), utilisent à fond deux technologies originales de programmation fonctionnelle, les contextes linéaires (ou *zippers*) permettant l'édition locale de structures mutables, et le foncteur de partage, permettant la minimisation de structures arborescentes générales. Une méthodologie uniforme d'implémentation de transducteurs par morphisme d'arbre lexical a été développée.

Ces techniques ont fait l'objet d'une présentation au Colloque en l'honneur de N. de Bruijn, *Workshop on Thirty Five Years of Automath*, tenu en avril 2002 à l'Université Heriot-Watt d'Edimbourg [15] ; d'un cours à l'Ecole d'été ESSLLI à Trento en août 2002 [23] ; d'un tutoriel à la conférence Language Engineering Conference LEC'2002 à Hyderabad en décembre 2002 [17] ; et d'une conférence invitée au Symposium *Practical Aspects of Declarative Programming* (PADL'03) en janvier 2003 à la Nouvelle Orléans [18]. La boîte à outils (appelée « Zen » pour souligner sa simplicité) est distribuée comme logiciel libre sur le site du projet (http://caml.inria.fr/humps/index.html).

7. Contrats industriels

7.1. Consortium Caml

Le Consortium Caml a pour vocation de réunir les industriels et académiques désireux d'aider au développement et de pérenniser le langage Caml et les outils associés. Cette structure est décrite plus en détails sur son site Web, http://caml.inria.fr/consortium/. À la suite d'une réunion préliminaire tenue à Rocquencourt au début juillet 2001, cinq membres industriels ont rejoint le Consortium en novembre 2001. Le Consortium a tenu deux assemblées dans le courant de l'année 2002. Au cours de la seconde, les formalité d'adhésion au Consortium ont été simplifiées et il a été décidé d'accorder une licence moins restrictive aux adhérents du Consortium. Ceux-ci pourront dorénavant embarquer par exemple une partie du compilateur OCaml dans des applications propriétaires.

8. Actions régionales, nationales et internationales

8.1. Actions nationales

8.1.1. ACI GRID

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

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

8.2. Actions financées par la Commission Européenne

8.2.1. Groupe de travail Esprit Applied Semantics

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

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

8.3. Relations bilatérales internationales

8.3.1. Inde

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

9. Diffusion des résultats

9.1. Animation de la communauté scientifique

9.1.1. Animation interne à l'INRIA

Gérard Huet a présidé la section locale d'audition du concours CR2 de l'UR Futurs.

Xavier Leroy a été (jusqu'en juillet 2002) membre titulaire élu de la Commission d'Évaluation de l'INRIA. À ce titre, il a fait partie du jury de recrutement CR1, du jury de recrutement DR2, et des sections locales d'audition CR2 des unités de Rennes et de Sophia, tout ceci ayant entraîné l'écriture de 17 rapports sur des candidatures.

Michel Mauny a été (jusqu'en juillet 2002) membre suppléant élu de la Commission d'Évaluation. Il a présidé la Section Locale d'Audition du concours CR2 2002 de l'UR de Rocquencourt et a participé aux jurys d'admissibilité de recrutement CR1, DR2 et DR1 de l'INRIA. Il est membre de l'équipe de direction de l'UR, et a été membre du groupe de travail sur le logiciel libre organisé par la DirDRI. Il a été membre de la commission de sélection des ARC (Actions de Recherche Coopérative) en décembre 2002.

Pierre Weis a été élu représentant des chercheurs (suppléant) à la Commission d'Évaluation.

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

Avec Michel Loyer pour la dimension administrative, Pierre Weis est co-responsable des moyens informatiques de Rocquencourt depuis mai 1999. Au sein de ce tandem, Pierre Weis s'occupe des bonnes relations avec les chercheurs : il recueille les besoins des projets et facilite l'établissement et le suivi des bons de commande. Cette année, Pierre Weis a notamment sélectionné et testé une méthode de déport des machines bruyantes reposant sur l'utilisation du câblage en paires torsadées existant.

9.1.2. Animation de la communauté scientifique hors INRIA

Gérard Huet est membre de *Academia Europaea* et correspondant de l'Académie des Sciences. Il a été élu membre de l'Académie des Sciences en novembre 2002. Il est membre du Board de l'Institut de Software Engineering de l'Université des Nations-Unies (UNU/IIST) à Macao.

Gérard Huet a participé en juillet 2002 à une mission d'évaluation des centres de mathématiques portugaises, organisée par la Science and Technology Foundation (FCT) du Ministère de la Recherche du Portugal. Il est responsable pour l'INRIA des relations scientifiques avec l'Inde.

Xavier Leroy est membre du groupe de travail IFIP 2.8 sur la programmation fonctionnelle. Il est membre du comité de pilotage du congrès *International Conference on Functional Programming*. Xavier Leroy est

aussi conseiller scientifique du projet européen IST « Dart » (Dynamic Assembly, Reconfiguration and Type-checking), et participe au réseau thématique IST « Reset » (Roadmaps for European research on Smartcard Technologies) dans la branche *Software and systems*, qui est sous la responsabilité de Gilles Barthe (INRIA Sophia-Antipolis).

Roberto Di Cosmo a co-organisé, avec Giuseppe Longo et Sergei Soloviev, le premier *workshop* international sur les isomorphismes de type (WIT'02). Cette manifestation s'est déroulée les 8 et 9 novembre 2002 à Toulouse et a réuni 24 chercheurs venus de toute l'Europe. Un numéro spécial d'une revue qui réunira les meilleurs travaux présentés dans ce colloque est en préparation.

Michel Mauny est membre du conseil scientifique du GDR *Algorithmique, Langage et Programmation* (ALP). Michel Mauny est membre du Comité Scientifique de la coopération franco-marocaine dans le domaine des STIC (programme géré par l'INRIA du côté français).

9.1.3. Animation de la communauté des usagers de Caml

Participants : Xavier Leroy, Maxence Guesdon, Pierre Weis.

9.1.3.1. Traduction collective

En collaboration avec Ruchira Datta (U. Berkeley), Joshua D. Guttman (MITRE Corp) et Benjamin C. Pierce (U. Penn), Xavier Leroy a supervisé l'effort collaboratif de traduction vers l'anglais du livre *Développement d'applications avec Objective Caml*, publié chez O'Reilly. Cette traduction a mobilisé 24 traducteurs volontaires et 37 relecteurs, interagissant via le réseau dans le plus pur esprit « logiciel libre ». Xavier Leroy a mis sur pieds les moyens informatiques nécessaires, et traduit deux chapitres. À l'INRIA, François Thomasset (projet A3) et Alan Schmitt (projet Moscova) ont également participé à la traduction et la relecture. Le résultat de la traduction est actuellement diffusé sur le Web, et des pourparlers sont en cours avec O'Reilly pour une édition sur papier. Il s'agit d'une étape importante dans la diffusion d'Objective Caml, car c'est le premier livre en anglais sur ce langage.

9.1.3.2. Le Caml Hump

Maxence Guesdon a généralisé les pages web du *Caml Hump*, pour en faire les *Caml Humps* regroupant les différents outils, applications, et bibliothèques relatifs à Caml et OCaml, répartis dans plusieurs thèmes (Caml/OCaml, OCamldoc, Camlp4, LablGtk et propositions de développement). Il a aussi ajouté la possibilité d'effectuer une recherche dans ces pages.

9.1.3.3. Formation

Pierre Weis continue à suivre et à encourager les efforts des professeurs des classes préparatoires. Il s'est rendu à Marseille au colloque de l'association des professeurs de mathématiques des classes préparatoires, du 2 au 5 mai 2002, où il a fait un cours d'introduction à l'utilisation d'Active-DVI.

9.1.4. Comités de lecture et programmes

Gérard Huet fait partie des comités de rédaction des revues *Journal of Symbolic Computation*, *Journal of Applied Non-Classical Logics*, *Journal of Logic and Computation*, *International Journal on Foundations of Computer Science* et *Annals of Pure and Applied Logic*. Il a été membre du comité de programme de la *Language Engineering Conference* (LEC'2002) qui s'est tenue à Hyderabad en décembre 2002.

Xavier Leroy est membre du comité de rédaction de la revue *Journal of Functional Programming*. Cette année, il a fait partie des comités de programmes du congrès *Smart Card Research and Advanced Application Conference* (CARDIS'02), et des ateliers *Security of Communications on the Internet '02* (SECI'02), *Types in Language Design and Implementation '03* (TLDI'03) *Foundations of Object-Oriented Languages* (FOOL'03), et Journées Francophones des Langages Applicatifs (JFLA'03).

Didier Rémy est membre du comité de rédaction de la revue *Journal of Functional Programming*. Il a aussi fait partie du comité de programme de l'*European Symposium On Programming (ESOP)* qui a eu lieu en avril 2002 à Grenoble.

François Pottier a été membre des comités de programme des conférences *Principles of Programming Languages* (POPL'03) et *Foundations of Software Science and Computation Structures* (FOSSACS'03).

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

9.1.5. Jurys de thèses

Gérard Huet a été rapporteur de l'habilitation de Christian Rétoré (université de Nantes, 4 jan. 2002) et a présidé le jury d'habilitation de Jean-François Monin (université Paris Sud, avril 2002).

Xavier Leroy a été rapporteur de la thèse d'Eva Rose (Paris 7, sept. 2002). Il a participé aux jurys d'habilitation de Renaud Rioboo (Paris 6, décembre 2002) et de thèse de Jun Furuse (Paris 7, décembre 2002).

Roberto Di Cosmo a été membre des jurys de thèse de Vincent Balat et Jean-Vincent Loddo (Paris 7, décembre 2002).

Michel Mauny a été membre des jurys de thèse de Pascal Cuoq (Paris 6, décembre 2002) et de Robert Harley (Paris 7, décembre 2002).

Pierre Weis a été rapporteur de la thèse de Bruno Barbier qui s'est déroulée le 20 novembre 2002 à l'université de Besançon, et membre du jury de la thèse de Jun Furuse (Paris 7, décembre 2002).

9.1.6. Autres activités d'intérêt général

9.1.6.1. Diffusion des connaissances

Participants : Roberto Di Cosmo, Pierre Weis.

Pierre Weis a réalisé une sixième version du CD-ROM de diffusion des logiciels de l'INRIA, « Logiciels libres disponibles à l'INRIA ». Ce CD-ROM comprend tous les logiciels du projet et ceux de tous les projets de l'INRIA qui ont répondu à cette initiative (plus de 30 logiciels).

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

La version 3.01pl5 a été éditée conjointement avec le CD des Logiciels Libres de l'Inria, et rendue disponible en octobre 2002 ; elle incorpore notamment deux des langages développés à l'INRIA : Ocaml (version 3.04) et Elan, ainsi que nos logiciels Active-DVI et Whizzytex.

9.2. Enseignement

9.2.1. Encadrement et jurys

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

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

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

François Pottier co-encadre avec Guy Cousineau la thèse de Vincent Simonet.

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

Pierre Weis a encadré la thèse de Jun Furuse.

9.2.2. Enseignements de troisième cycle universitaire

Roberto Di Cosmo est le responsable du DEA « Programmation : sémantique, preuves et langages » de l'Université Paris 7 (dénommé « DEA SPP » par la suite). Il enseigne avec Delia Kesner le cours d'option « Logique Linéaire et Calcul des Motifs » de ce même DEA.

Xavier Leroy est responsable de la filière « Langages et applications » du DEA SPP.

Xavier Leroy a donné deux cours de 5h sur la compilation des langages fonctionnels, l'un à l'école de printemps « Sémantique des langages de programmation » organisée à Agay par le laboratoire PPS de l'université Paris 7, l'autre aux élèves de 3ième année de l'ENS Cachan.

Gérard Huet a dispensé un cours de 10h à l'École d'été ESSLLI à Trento en Italie, où il a présenté *The Zen Computational Linguistics Toolkit*.

Gérard Huet a donné 4h de cours en tronc commun et 12h d'option « Traitement informatique de la langue » au DEA d'Informatique de l'Université Bordeaux 1.

François Pottier a donné un cours de 3 heures dans le cadre de l'École de Printemps organisée à Agay en mars 2002 par le laboratoire PPS de l'Université Paris 7.

François Pottier a assuré le cours de tronc commun (20h) *Typage et programmation*, dans le cadre du DEA SPP.

Didier Rémy a publié cette année ses notes de cours pour l'École d'été APPSEM qui a eu lieu en septembre 2000 à Camiña, Portugal dans un recueil dédié [9].

9.2.3. Enseignement en écoles d'ingénieurs

Didier Rémy est professeur chargé de cours à temps partiel à l'École Polytechnique.

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

Didier Le Botlan a assuré un total de 40 heures à l'École Polytechnique en TP de Java et correction de travaux informatiques.

9.2.4. Autres enseignements

Michel Mauny et Emmanuel Chailloux (Paris 6, laboratoire PPS) ont dispensé une formation de deux jours à Objective Caml dans le cadre de tutoriels préliminaires à la conférence CARI'02, organisée à Yaoundé en octobre 2002. Les auditeurs étaient des étudiants, chercheurs et enseignants africains, venus du Cameroun et des pays voisins.

9.3. Participation à des colloques, séminaires, invitations

Gérard Huet a présenté une conférence invitée *Higher Order Unification 30 years later* à la Conférence Internationale TPHOL 2002 à Hampton (Virginia) en juillet 2002.

Daniel Bonniot, François Pottier et Vincent Simonet ont assisté à la conférence POPL'02 puis au workshop FOOL qui se sont tenus à Portland (Oregon) en janvier 2002. Vincent Simonet y a présenté son travail commun avec François Pottier concernant l'inférence de flots d'information pour ML. Daniel Bonniot a présenté à FOOL son article sur le typage des multi-méthodes [11].

Benjamin Grégoire, Xavier Leroy, Michel Mauny et Didier Rémy ont assisté au congrès *International Conference on Functional Programming* (Pittsburgh, septembre 2002). Xavier Leroy a présenté son travail commun avec Benjamin Grégoire sur la compilation de la réduction forte. Michel Mauny a assisté au *workshop* PLAN-X qui a eu lieu à cette occasion.

Tom Hirschowitz, Xavier Leroy et Didier Rémy ont assisté à la conférence ESOP'02 à Grenoble. Tom Hircshowitz y a présenté son travail avec Xavier Leroy sur les modules mixins [14].

François Pottier et Vincent Simonet ont assisté au workshop CSFW'15 à Cape Breton (Canada) en juin 2002, où chacun d'eux a présenté un article concernant l'inférence de flots d'informations [19][21].

Michel Mauny a participé à la conférence *Mathematics of Program Construction* (MPC'02), à la *IFIP WG 2.1 Working Conference on Generic Programming* et au *Workshop on Types in Programming* (TIP'02) qui ont eu lieu à Dagstuhl en juillet 2002. Daniel Bonniot a participé à TIP'02, où il a présenté son article [12].

Daniel Bonniot, Tom Hirschowitz et Vincent Simonet ont assisté à l'école de printemps *Sémantique des langages de programmation* organisée à Agay (Var) en avril 2002 par le laboratoire Preuves, Programmes et Systèmes de l'Université Paris 7.

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

Vincent Simonet à présenté ses travaux au séminaire du Magistère d'Informatique et de modélisation de l'École Normale Supérieure de Lyon (octobre) puis au séminaire des élèves du département d'Informatique de

l'École Normale Supérieure de Paris (novembre). Il a également donné un exposé au séminaire du laboratoire Preuves, Programmes et Systèmes de l'Université Paris 7 (décembre).

François Pottier a assisté à une réunion du groupe Profundis à Sophia Antipolis (avril 2002), et y a présenté un exposé concernant l'inférence de flots d'information dans le π-calcul.

Tom Hirschowitz a passé une semaine à l'université de Heriot-Watt à Edimbourg (Royaume Uni), dans le cadre de sa collaboration avec Joe B. Wells sur les modules mixins.

François Pottier a rendu une visite de quelques jours aux membres du Computer Science Lab de l'université Cornell, à Ithaca (septembre 2002).

9.4. Relations industrielles

Gérard Huet est membre de l'*advisory committee of TECS for Tata Consultancy Services*.

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

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

10. Bibliographie

Thèses et habilitations à diriger des recherche

[1] P. CUOQ. *Ajout de synchronisme dans les langages fonctionnels fortement typés.* thèse de doctorat, Université Paris 6, 2002.

[2] J. FURUSE. *Extensional polymorphism in ML.* thèse de doctorat, Université Paris 7, 2002.

[3] R. HARLEY. *Advanced Algorithms for Arithmetic on Curves.* thèse de doctorat, Université Paris 7, 2002.

Articles et chapitres de livre

[4] J.-C. FILLIÂTRE, F. POTTIER. *Producing All Ideals of a Forest, Functionally.* octobre, 2002, http://pauillac.inria.fr/~fpottier/publis/filliatre-pottier.ps.gz, À paraître dans le *Journal of Functional Programming*.

[5] G. HUET. *Śrī Yantra Geometry.* in « Theoretical Computer Science », volume 281, 2002, pages 609-628.

[6] G. HUET. *Linear Contexts and the Sharing Functor : Techniques for Symbolic Computation.* in « Special volume in honor of N.G. de Bruijn », 2003, à paraître.

[7] X. LEROY. *Bytecode Verification for Java Smart Card.* in « Software Practice & Experience », volume 32, 2002, pages 319-340.

[8] F. POTTIER, V. SIMONET. *Information Flow Inference for ML.* août, 2002, http://pauillac.inria.fr/~fpottier/publis/fpottier-simonet-toplas.ps.gz, À paraître dans *ACM Transactions on Programming Languages and Systems*.

[9] D. RÉMY. *Using, Understanding, and Unraveling the OCaml Language.* éditeurs G. BARTHE., in « Applied Semantics. Advanced Lectures. LNCS 2395. », Springer Verlag, 2002, pages 413-537.

Communications à des congrès, colloques, etc.

[10] V. BALAT, R. DI COSMO, M. FIORE. *Remarks on Isomorphisms in Typed Lambda Calculi with Empty and Sum Type.* in « Logic in Computer Science », IEEE Press, juillet, 2002, http://www.cl.cam.ac.uk/~mpf23/papers/Types/remarks.ps.gz.

[11] D. BONNIOT. *Type-checking multi-methods in ML (A modular approach).* in « The Ninth International Workshop on Foundations of Object-Oriented Languages, FOOL 9 », Portland, Oregon, USA, janvier, 2002, http://cristal.inria.fr/~bonniot/bonniot02.ps.

[12] D. BONNIOT. *Using kinds to type partially polymorphic multi-methods.* in « Workshop on Types in Programming (TIP'02) », Dagstuhl, Germany, juillet, 2002, http://cristal.inria.fr/~bonniot/bonniot02using.ps.

[13] B. GRÉGOIRE, X. LEROY. *A compiled implementation of strong reduction.* in « International Conference on Functional Programming 2002 », ACM Press, pages 235-246, 2002.

[14] T. HIRSCHOWITZ, X. LEROY. *Mixin modules in a call-by-value setting.* in « Programming Languages and Systems, ESOP'2002 », série Lecture Notes in Computer Science, volume 2305, éditeurs D. LE MÉTAYER., pages 6-20, 2002.

[15] G. HUET. *Data Structures for Trees and Tree Contexts, and a Uniform Sharing Functor, as Design Issues for Symbolic Computation Systems.* in « Workshop on Thirty Five Years of Automath », Heriot-Watt University, Edinburgh, 2002, http://pauillac.inria.fr/~huet/PUBLIC/DB.ps.

[16] G. HUET. *Higher Order Unification 30 YEARs later.* in « Proceedings, 15th International Conference TPHOL 2002 », Springer-Veerlag LNCS 2410, éditeurs C. M. V. CARREÑO, S. TAHAR., pages 3-12, 2002.

[17] G. HUET. *Lexicon and Morphotactics : The Zen Computational Linguistics Toolkit.* in « Tutorial, Language Engineering Conference LEC'2002, Hyderabad (Andhra Pradesh, India) », décembre, 2002.

[18] G. HUET. *Zen and the Art of Symbolic Computing : Light and Fast Applicative Algorithms for Computational Linguistics.* in « Symposium on Practical Aspects of Declarative Programming (PADL'03), New Orleans (Louisiana) », janvier, 2003.

[19] F. POTTIER. *A Simple View of Type-Secure Information Flow in the π-Calculus.* in « Proceedings of the 15th IEEE Computer Security Foundations Workshop », pages 320-330, Cape Breton, Nova Scotia, juin, 2002, http://pauillac.inria.fr/~fpottier/publis/fpottier-csfw15.ps.gz.

[20] F. POTTIER, V. SIMONET. *Information Flow Inference for ML.* in « Proceedings of the 29th ACM Symposium on Principles of Programming Languages (POPL'02) », pages 319-330, Portland, Oregon, janvier, 2002, http://pauillac.inria.fr/~fpottier/publis/fpottier-simonet-popl02.ps.gz.

[21] V. SIMONET. *Fine-grained Information Flow Analysis for a λ-calculus with Sum Types.* in « Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW 15) », pages 223-237, Cape Breton, Nova Scotia (Canada), juin, 2002, http://cristal.inria.fr/~simonet/publis/simonet-csfw-02.ps.gz.

[22] C. SKALKA, F. POTTIER. *Syntactic Type Soundness for $HM(X)$.* in « Proceedings of the 2002 Workshop on Types in Programming (TIP'02) », Dagstuhl, Germany, juillet, 2002, http://pauillac.inria.fr/~fpottier/publis/skalka-fpottier-tip-02.ps.gz.

Rapports de recherche et publications internes

[23] G. HUET. *The Zen Computational Linguistics Toolkit.* rapport technique, ESSLLI Course Notes, 2002, http://pauillac.inria.fr/~huet/ZEN/index.html.

[24] X. LEROY, D. RÉMY, J. GARRIGUE, J. VOUILLON, D. DOLIGEZ. *The Objective Caml system, documentation and user's manual - release 3.06.* INRIA, août, 2002, http://caml.inria.fr/ocaml/htmlman/, documentation distribuée avec le système Objective Caml.

Divers

[25] T. HIRSCHOWITZ, X. LEROY. *Mixin modules in a call-by-value setting.* 2002, Soumis à publication aux *ACM Transactions on Programming Languages and Systems*.

[26] G. HUET. *Transducers as Lexicon Morphisms, Segmentation by Euphony Analysis, And Application to a Sanskrit Tagger.* 2002, http://pauillac.inria.fr/~huet/FREE/tagger.ps.

[27] X. LEROY. *Java bytecode verification : algorithms and formalizations.* 2002, Soumis à publication au *Journal of Automated Reasoning*.

[28] V. SIMONET. *Type inference with structural subtyping : A faithful formalization of an efficient constraint solver.* octobre, 2002, http://cristal.inria.fr/~simonet/publis/simonet-structural-subtyping.ps.gz, Soumis à publication.

Bibliographie générale

[29] 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, pages 113-123, 1993.

[30] J. GARRIGUE, D. RÉMY. *Extending ML with Semi-Explicit Higher-Order Polymorphism.* in « Journal of Functional Programming », numéro 1/2, volume 155, 1999, pages 134-169, ftp://ftp.inria.fr/INRIA/Projects/cristal/Didier.Remy/iandc.ps.gz, une version préliminaire a été présentée à TACS'97.

[31] X. LEROY. *A syntactic theory of type generativity and sharing.* in « Journal of Functional Programming », numéro 5, volume 6, 1996, pages 667-698.

[32] X. LEROY, P. WEIS. *Manuel de référence du langage Caml.* InterÉditions, juillet, 1993.

[33] D. RÉMY. *Programming Objects with ML-ART : An extension to ML with Abstract and Record Types.* in « Theoretical Aspects of Computer Software », série Lecture Notes in Computer Science, volume 789, Springer-Verlag, éditeurs M. HAGIYA, J. C. MITCHELL., pages 321-346, avril, 1994.

[34] P. WEIS, X. LEROY. *Le langage Caml.* Dunod, juillet, 1999, Deuxième édition.