INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet Moscova

Mobilité, Sécurité, Concurrence, Vérification et Analyse

Rocquencourt

THÈME 1C

Table des matières

1. Composition de l'équipe

Responsable scientifique
Georges Gonthier [DR]

Responsable permanent
Luc Maranget [CR]

Assistante de projet
Sylvie Loubressac [TR]

Personnel Inria
Damien Doligez [CR]
Thérèse Hardin [en délégation de Paris VI, jusqu'au 01/09/2002]
James Leifer [CR au 01/09/2002]
Jean-Jacques Lévy [DR]

Conseiller extérieur
Thérèse Hardin [depuis le 01/09/2002]

Post-Doctorant
James Leifer [jusqu'au 01/09/2002]

Doctorants
Gilles Peskine [allocataire MESR, AMN]
Ma Qin [allocataire MENRT, Paris VII]
Virgile Prévosto [allocataire DGA, Paris VI]
Alan Schmitt [boursier DGA, jusqu'au 01/10/2002]

Stagiaire
SriRam Thirthala Venkata [ITT, New-Dehli]

2. Présentation et objectifs généraux

Le projet Moscova s'intéresse à la programmation concurrente, à la mobilité comme primitive de programmation, à la sécurité, et aux preuves formelles. Le projet participe aussi, mais marginalement, à l'effort de programmation autour du langage Caml.

La programmation de plusieurs processus concurrents est délicate. Elle exige de bien comprendre le modèle sous-jacent et de disposer de primitives rigoureusement définies. Notre objectif est de construire des systèmes pour programmer des applications faiblement synchrones sur des architectures distribuées. Dans ce cadre, notre effort se dirige actuellement vers la programmation de processus mobiles qui permettent de tenir compte de la reconfiguration dynamique des interconnexions.

Il nous semble que les applications distribuées à grande échelle vont avoir tendance à se développer, avec des serveurs multi-sites qui s'envoient des informations de haut niveau pour contourner les limitations de bande passante. Les données et les programmes vont s'échanger avec les problèmes de sécurité conséquents. Les appels de procédures distantes seront remplacés par l'envoi d'*agents mobiles*. Une communauté de langages de programmation avec agents mobiles existe : Facile [31], de B. Thomsen à l'ECRC-Munich, Obliq de L. Cardelli à DEC/SRC, Pict [30] de B. Pierce et D. Turner à Edimbourg/Indiana/Penn, Nomadic-Pict de P. Sewell et P. Wojciechowski à Cambridge. D'autres langages, comme Java, permettent le chargement de petits programmes (les *applets*) sur le *Web*. Parmi ces propositions, seuls Obliq et Nomadic-Pict permettent l'envoi de sous-programmes actifs, c'est-à-dire de processus avec leurs environnements actifs en cours d'exécution. En travaillant sur la définition d'un Pict distribué, nous avons introduit en 1995 un nouveau calcul, le *join-calcul*, dont nous avons montré la relation avec le π-calcul [32] de Milner, base théorique du langage Pict. Ce nouveau calcul est implémentable dans un environnement distribué. Nous avons deux

systèmes disponibles en *ftp* anonyme : le join-calcul 1.05 et *JoCaml* complètement compatible avec le langage OCaml.

L'intérêt de notre projet est porté non seulement vers les fondements sémantiques des langages de programmation, mais aussi vers leur implémentation. Nous avons des relations techniques suivies avec d'autres groupes de l'INRIA travaillant dans les domaines des systèmes d'exploitation et de la compilation des langages de programmation. Notre projet a une tradition de participation au développement de Caml (filtrage, glaneur de cellules) et contribue aux versions successives. Le pari fait avec JoCaml est d'avoir aussi une version distribuée et mobile du système Caml. Cette dernière partie a pris du retard en 2002 après le départ de F. le Fessant. Elle constitue néanmoins un des objectifs majeurs du projet Moscova.

Du coté théorique, nous poursuivons des actions sur les équivalences de processus, les protocoles de sécurité, sur l'analyse de flots, sur le contrôle de ressources, sur les objets concurrents. Le travail sur le join-calcul a été effectué par la grande majorité des membres de notre projet et par D. Rémy du projet Cristal. En 2002, nous avons commencé une nouvelle collaboration avec Cambridge, l'EPFL, SICS, KTH et UCL sur les applications distribuées symétriques, *peer to peer*, dans le cadre du projet Pepito faisant partie de l'action *Global Computing* de l'Union Européenne.

Par ailleurs, nous n'avons plus d'activité sur l'analyse statique de programmes, après les départs successifs de A. Deutsch pour Polyspace Tech. et de B. Blanchet pour le CNRS. Le contrat avec la société danoise Terma sur la robustesse de l'informatique embarquée pour les applications spatiales pour le compte de l'ESA, s'est également terminé au début de l'année 2002.

Depuis deux ans, notre projet a une activité dans les preuves formelles. Cette activité pré-existait avec la longue preuve de sûreté du glaneur de cellules de Caml concurrent (dont la version mono-processeur est le glaneur de cellules incrémental du système OCaml). Elle s'est renforçée avec l'accueil en délégation de Thérèse Hardin. A présent, nous sommes fortement impliqués dans le projet FoC de calcul formel certifié qu'elle dirige ; Damien Doligez et Virgile Prévosto y participent principalement. Par ailleurs, Georges Gonthier et Benjamin Werner (projet Logical) se sont également lancé dans une longue preuve mécanique du théorème des quatre couleurs pour la coloration des graphes planaires dans le système Coq.

En 2002, Sylvain Conchon et Alan Schmitt ont passé leur thèses (sous la direction de J.-J. Lévy), et sont partis en année post-doctorale repectivement à Oregon Graduate Institute et à l'Université de Pennsylvanie. James Leifer a été embauché comme Chargé de Recherche. Cette année a également été marquée par une forte activité d'enseignement à l'Ecole polytechnique de plusieurs membres de notre projet, suite à la mise en place du nouveau cursus X2000.

3. Fondements scientifiques

3.1. Le join-calcul

R. Milner a démarré la théorie de la concurrence en 1980 à Edimbourg, en proposant un calcul des systèmes communicants (CCS)[26]. Cette théorie a eu de nombreux développements algébriques ou logiques, qui ont démontré le coté non trivial de sa modélisation. En 1989, R. Milner, J. Parrow et D. Walker[28] ont introduit un nouveau calcul, le *pi-calcul*, permettant de considérer les réseaux de communication reconfigurables. Cette théorie a été reprise puis affinée par D. Sangiorgi (Edimbourg/INRIA Sophia). Beaucoup d'autres calculs de processus ont fleuri pendant ces 15 dernières années. Notre projet s'intéresse principalement au pi-calcul dont la théorie de l'équivalence est loin d'être simple : bisimulations *early*, *late*, *open*, etc.

Nous avons introduit un nouveau calcul de processus, le join-calcul [5][6], qui est facilement implémentable dans tout système distribué, car il ne nécessite aucun protocole de communication sophistiqué, tels que ceux qui permettent la diffusion atomique ou le consensus distribué.

Le join-calcul permet la programmation concurrente et distribuée, ainsi qu'une communication et une coopération simple entre deux tâches qui s'exécutent sur des machines différentes. Dès le départ, le join-calcul a été conçu en tenant compte de la « localisation » des objets manipulés (processus, canaux, groupes de tels objets). Ce souci a facilité la production d'un langage de programmation qui donne au programmeur

une vision de relativement haut niveau d'un réseau de machines. Classiquement, cette vision de haut niveau cache les détails de la programmation distribuée sur un système particulier et permet au programmeur de se concentrer sur l'architecture de son programme.

Dans le cas du join-calcul, une première brique est la communication par canaux, qui peuvent eux-mêmes voyager sur les canaux. Les canaux relient des processus entre eux, une restriction fondamentale du join-calcul est d'imposer un récepteur unique sur chaque canal, ce qui permet l'identification d'un canal avec son récepteur. La perte d'expressivité qui en résulte est compensée par la réception jointe, une certaine action ne se déclenchant que si plusieurs canaux reçoivent un message. La nature du calcul et du langage permet la localisation des canaux engagés dans une réception jointe sur la même machine. La réalisation de rendez-vous entre des processus réellement distribués (i.e., résidant sur des machines différentes) devient possible car ce rendez-vous aura lieu par une communication jointe qui se décomposera en deux phases, une phase de transport des messages vers la machine qui possède les canaux de la réception jointe, puis une phase de rendez-vous purement locale et donc assez simple.

La deuxième brique de base du join-calcul est la location. Le contrôle effectif de la localisation des canaux et processus se fait par ce biais : une location est un ensemble de canaux et de processus conçus pour migrer ensemble d'une machine à l'autre. Les locations pouvant être créées dynamiquement, il s'ensuit une notion naturelle de parenté entre locations, puis de structure arborescente des locations. La migration d'une location correspond au déplacement d'un sous-arbre vers un autre point d'attache dans l'arbre des locations. Ici encore, le langage reprend un concept issu des calculs de processus, les machines n'apparaissent pas dans le langage, seul apparaît une location racine par machine. Le réseau de machine est donc abstrait en une forêt de locations, les migrations sont explicites, dans le sens que le programmeur spécifie bien une migration, mais elles restent d'assez haut niveau, tant par ce qui migre (un ensemble cohérent de canaux et de processus contenu dans une location et toutes ses sous-locations) que par la façon dont est donnée la destination de la migration (une autre location dont la location migrante devient une sous-location). Cette approche est entièrement intégrée dans notre prototype et permet l'écriture de programmes distribués, avec communications distantes, migration de code, applets, etc.

3.2. La sémantique des langages de programmation et leur compilation

Notre projet s'attache à développer des compilateurs pour des langages directement inspirés du join-calcul. Il y a là, à notre avis, un peu plus qu'une simple démonstration de faisabilité. Rappelons en effet que le join-calcul est dès le départ conçu en vue d'une réalisation authentiquement concurrente et distribuée, c'est-à-dire s'exécutant effectivement sur plusieures machines connectées. Un telle démarche n'est complète que si un langage est effectivement produit.

Dans une premier temps nous avions développé le langage `join-calcul` qui est bien un prototype. Notre deuxième langage `jocaml`, est plus ambiteux, car il s'agit d'une extension de Objective Caml. Nous dépassons ainsi le stade du prototype car `jocaml` (disonible depuis 1998) propose d'emblée toutes les constructuctions d'Objective Caml qui est un langage de programmation déjà mature. Notre effort peut alors se porter sur les traits concurrents que nous ajoutons, mais nous devons alors tenir compte de la sophistication et surtout l'évolution d'Objective Caml lui-même. Or le `jocaml` disponible a été conçu comme une extension d'une version donnée de Objective Caml, ajourd'hui largement obsolete. Notre effort porte donc maintenant sur l'intégration complète de `jocaml` dans Objective Caml. Plus précisément il y aura un compilateur commun à Caml et à `jocaml`.

Cette intégration est d'autant plus logique et possible que plusieurs membres de notre projet participent déjà activement au dévelopememnt d'Objective Caml (Damien Doligez pour le glaneur de cellules, Luc Maranget pour le filtrage). Par ailleurs le développement d'un langage concurent soulève des questions qui se posent également dans le cadre du développement de Objective Caml lui-même. On peut citer l'integration du typage dynamique de valeurs persistantes seulement en Caml (et distribués en `jocaml`) dans le système de type essentiellement statique de Caml. On peut aussi penser à la réalisation d'une interface uniforme et régulière

pour intégrer de nouveaux type de données « primitifs » dans Caml, dont les principes ont été posés par Fabrice Le Fessant dans le premier `jocaml` et sont d'ores et déjà présents en Caml.

4. Domaines d'application

Mots clés : *télécommunication*, *application répartie*, *sécurité*, *vérification*.

La programmation distribuée avec mobilité intervient dans la programmation du World Wide Web et des systèmes mobiles autonomes. Elle permet la personnalisation des interfaces et de la communication entre plusieurs clients. Les télécommunications sont un autre domaine d'application, avec les commutateurs actifs ou les services dits intelligents.

5. Logiciels

5.1. JoCaml

Participants : Fabrice le Fessant, Luc Maranget, Alan Schmitt.

Ce logiciel est disponible sur le réseau en http://pauillac.inria.fr/jocaml.

Le travail en cours sur JoCaml consiste en la production d'une version stable, fortement liée à Caml lui-même. La technique retenue est l'intégration complète de JoCaml dans Caml. Ainsi, on aboutit à un important partage de code *source* avec Caml pour tout ce qui ne concerne pas la concurrence. La maintenance de JoCaml s'en trouvera grandement facilitée, notamment en ce qui concerne le suivi de l'évolution de Caml, qui reste soutenue.

Un première étape, intégrer le compilateur JoCaml dans le compilateur Caml, s'est révélée plus facile que prévu. Les ajouts au compilateur Caml sont en effet minimaux. Outre, une extension d'analyseur syntaxique (limitée à 170 lignes grâce à Camlp4), le prototype de compilateur JoCaml ne diffère que d'environ 2000 lignes du compilateur Caml d'origine. Par manque d'un environnement d'exécution adéquat, ces extensions sont encore très peu testées et on doit donc les considérer comme faisant partie d'un premier jet.

La seconde étape, la production d'un environnement d'exécution distribué se révèle difficile. À l'imitation de l'ancien JoCaml, cet environnement doit comprendre entre autres, la gestion des processus légers, la réalisation des protocoles de communication et de migration, ainsi qu'un glaneur de cellules distribué. Toutes ces questions sont délicates, et ne sont pas, loin de là, des extensions faciles de de la machine abstraite Caml gérant les processus légers, qui est notre base de départ. Cet environnement d'exécution avait été réalisé dans la version précédente de Jocaml par F. le Fessant.

5.2. Le join-calcul 1.05

Participant : Luc Maranget.

Ce logiciel est disponible sur le réseau en http://join.inria.fr.

Ce système, toujours disponible, ne connait plus d'évolution. L'ensemble de nos efforts de développement se concentrant sur JoCaml. Outre son intérêt historique, cette première incarnation du join-calcul en tant que langage de programmation a fourni la base de la documentation de JoCaml.

5.3. Participation au développement d'Objective Caml

Participants : Damien Doligez, Luc Maranget.

D. Doligez et L. Maranget assurent la maintenance des parties d'Objective Caml qu'ils ont écrit (Glanage de cellules (GC) et compilation du filtrage).

D. Doligez a porté Objective Caml sur le système d'exploitation Mac OS X.

D. Doligez a implémenté l'optimisation des valeurs paresseuses dans Objective Caml. Une valeur paresseuse est un calcul « suspendu » qui ne sera effectué que si on a réellement besoin de son résultat.

Les valeurs paresseuses sont utilisées par exemple pour représenter des listes infinies, des développements en séries entières, etc. Une valeur paresseuse est représentée par une cellule d'indirection qui contient un pointeur, soit vers la fermeture qui représente le calcul suspendu, soit vers le résultat de ce calcul, quand il a été effectué (ce qui évite de le recalculer à chaque fois qu'on l'utilise). L'optimisation consiste a supprimer la cellule d'indirection et à remplacer le pointeur vers cette cellule par un pointeur direct vers la valeur calculée. Comme le GC examine régulièrement tous les pointeurs des structures de données du programme, il est naturel d'ajouter du code dans le GC pour effectuer ce remplacement de pointeurs. Il a fallu résoudre quelques problèmes non triviaux, notamment quand la valeur calculée est elle-même une valeur paresseuse. Cette optimisation permet à Objective Caml d'obtenir sur les paresseux des performances comparables à celles des systèmes Haskell, qui sont spécialisés dans l'évaluation paresseuse.

L. Maranget a réécrit et perfectionné le vérificateur du filtrage d'Objective Caml. Le filtrage des valeurs est une construction cruciale des langages de programmation fonctionnels. Il permet de discriminer entre plusieurs motifs représentant diverses valeurs algébriques possibles, typiquement des préfixes d'arbres, dans un type de données disjonctif. Cette construction qui s'apparente à l'analyse cas-par-cas, appelle deux contrôles élémentaires : tous les cas sont-ils traités ? tous les cas traités sont-ils utiles ? En termes de filtrage, on parle de vérification d'exhaustivité et de détection de clauses inutiles. Le nouvel analyseur est mieux compris, notamment du point de vue de l'explosion du temps de calcul, maîtrisée en pratique, et de ses rapports avec le typage des variants polymorphes. En outre le nouvel analyseur produit un avertissement spécifique dans le cas de motifs-ou (c'est-à-dire avec la disjonction à l'intérieur des motifs) partiellement inutiles. Ce travail a donné lieu à une publication aux Journées Francophones des Langages Applicatifs 2003 [15].

L. Maranget a travaillé sur l'extension de l'expressibilité des méta-analyseurs lexicaux. Le système Objective Caml comprend le générateur *ocamllex* d'analyseurs lexicaux du style de *lex*. Ce générateur transforme une suite de paires composées d'une expression régulière et d'une action en un analyseur lexical. Les expressions régulières peuvent maintenant contenir une nouvelle primitive de liaison p `as` x. Sans augmenter la classe des langages reconnus par les expressions régulières, cette nouvelle construction facilite l'écriture des analyseurs. Par exemple si on veut reconnaître deux entiers séparés par un ou plusieurs espaces, on écrira : `(['0'-'9']+ as n1) ' '+ (['0'-'9']+ as n2)`, et, dans l'action associée, on peut utiliser les variables `n1` et `n2` liées aux deux suites de chiffres reconnues. Sans les liaisons `as`, il faut reparcourir, après reconnaissance, la chaîne filtrée afin d'identifier les espaces, ou séparer la reconnaissance des deux entiers. La réalisation suit la technique classique de compilation des expressions régulières vers des automates finis déterministes. L'extension de cette technique aux constructions de liaison des sous-chaînes filtrées n'a suscité de l'intérêt que récemment, alors que les réalisations du filtrage par les expressions régulières selon une technique de *backtracking* proposent des constructions similaires de parenthèses liantes depuis longtemps. Le nouvel *ocamllex* est, à notre connaissance, le seul générateur d'analyseurs lexicaux proposant ce trait. L'extension n'est en effet pas immédiate, puisque le nouvel *ocamllex* comprend environ 2000 lignes nouvelles ou modifiées de Caml (sur environ 2700 en tout), plus environ 100 lignes de C.

Dans un domaine connexe, L. Maranget a intensivement testé la nouvelle réalisation de la bibliothèque d'expressions régulières standard de Objective Caml, ré-implémentée en décembre 2002 par Xavier Leroy. Il a trouvé de nombreuses erreurs, maintenant corrigées.

5.4. Hevea

Participant : Luc Maranget.

Ce logiciel est disponible sur le réseau en http://pauillac.inria.fr/~maranget/hevea/

Hévéa est un traducteur rapide de LaTeX vers HTML. Il est écrit en Objective Caml. Il date de 1997 et est maintenu et amélioré depuis par Luc Maranget. Hévéa donne lieu a une collaboration continue informelle avec Philip H. Viton (Ohio State University) pour le portage sur Windows et avec Ralf Treinen (ENS Cachan) pour le suivi Debian. Pour mémoire, Hévéa comprend environ 20000 lignes de Caml et 5000 lignes de packages écrites dans une syntaxe compatible avec celle de TeX. Nous ne disposons plus de statistiques complètes de

téléchargement. À titre indicatif, il y a eu environ 400 téléchargements de la version 1.06 (datée de mai 2002) entre le 2 décembre 2002 et le 6 janvier 2003.

5.5. Le système FOC

Participants : Damien Doligez, Thérèse Hardin, Virgile Prevosto.

Ce logiciel sera disponible au printemps 2003.

FOC est un environnement de programmation certifiée pour le calcul formel (voir section 6.11). Le logiciel regroupe le compilateur FOC (20000 lignes de code OCaml), développé principalement par V. Prévosto, la librairie de calcul formel (6000 lignes de code FOC), développée principalement par Renaud Rioboo (LIP6), et divers outils annexes (6000 lignes de code).

6. Résultats nouveaux

6.1. Le join-calcul

Participant : Georges Gonthier.

Un article de synthèse de G. Gonthier avec C. Fournet (Microsoft Research, Cambridge) [10], et un article de revue avec C. Fournet et M. Abadi (UC Santa Cruz) [9], ont finalement paru en 2002.

6.2. Dérivation de congruences opérationelles pour les calculs de processus

Participant : James Leifer.

James Leifer a travaillé sur deux publications tirées des recherches effectuées pour sa thèse de doctorat.

Ce travail s'attaque à la transformation d'un calcul de processus défini par des règles de réactions en un calcul défini par un système de transitions étiquetées (STE) raisonnable ; c'est depuis longtemps un problème ouvert de sémantique dans la théorie de la concurrence. En effet, si l'usage des règles de réaction s'est généralisé à cause de leur expressivité, le passage - difficile - aux STE reste nécessaire, entre autres pour la réalisation d'outils de vérification. Et si les STE permettent une définition naturelle de l'équivalence comportementale, la preuve que celle-ci est une congruence pour les opérateurs du calcul est en général non-triviale.

Ce travail amène trois contributions :

1. Il définit une méthode de *synthèse* de STE qui s'applique à une large classe de système réactifs ; elle consistà isoler des *étiquette contextuelles* qui sont « juste assez grandes » pour permettre une réaction.
2. Il formalise précisément cette notion de « juste assez grand » à l'aide de la notion catégorique de *somme amalgamée relative* (SAR, *relative pushout* en anglais).
3. Il démontre que, si les SAR existent suffisament, la bisimulation du STE ainsi obtenu est bien une congruence.

Les SAR peuvent ne pas exister pour les calculs dont la syntaxe est insuffisament structurée. Dans [19], Leifer introduit les *systèmes reactifs fonctoriels* pour pallier à ce cas, en s'appuyant sur une généralisation de la théorie existante. Il démontre en outre la congruence des *failures preorder* de Hoare [25], illustrant ainsi que la théorie des SAR peut s'utiliser aussi bien pour les équivalences inductives que coinductives.

Dans [20], Leifer définit une classe très générale de foncteurs qui satisfont les exigences des SAR ; il propose ensuite un exemple de système reactif fonctoriel, tiré de la théorie des *action calculi* de Milner [27], qui satisfait toutes les exigences de SAR requises dans les preuves de [19].

Les deux articles ont été soumis à *Mathematical Structures in Computer Science*.

6.3. Le join-calcul dynamique

Participant : Alan Schmitt.

Les programmes mobiles ont souvent besoin d'ajuster leur comportement à leur localisation courante, ne serait-ce que pour accéder à des ressources locales. Le système JoCaml offre plusieurs solutions à ce problème (utilisation de modules non migrants mais liés localement, serveur de noms locaux), mais ces solutions ne sont pas présentes dans le modèle théorique. De plus, l'utilisation de cette liaison dynamique aux ressources locales reste assez difficile à mettre en oeuvre pour le programmeur. L'approche proposée consiste à introduire de nouveaux récepteurs de canaux *dynamiques* vers lesquels sont dirigés les messages en fonction de la localisation de l'émetteur. Le récepteur est toujours trouvé sur la plus petite localité (au sens du join calcul) contenant une définission de canal dynamique. Cette notion peut donc changer en fonction de la localisation de l'émetteur. Un système de types (statiques) permet de garantir l'existence d'un récepteur pour tout canal dynamique existe quelle que soit la localisation de l'émetteur. Ainsi, aucun message ne peut être perdu.

Ce travail a été présenté à la conférence IFIP-TCS, Montréal, Canada[16]. Il fait aussi partie de la thèse d'A. Schmitt [8].

6.4. Contrôle de migration

Participant : Alan Schmitt.

Le M-Calcul est un calcul de processus cherchant à intégrer la communication du Join Calcul (transparente, efficace, facilement implémentable) au calcul des Ambients (possédant un contrôle très strict de la migration). Ce calcul est développé en collaboration avec J.-B. Stéfani de l'INRIA Rhône-Alpes, et est le prolongement du κ calcul développé dans le cadre du projet RNRT Marvel.

Cette année, le calcul a été finalisé et un système de types garantissant l'unicité des noms de chaque localité active a été prouvé correct. De nombreuses extensions sont en cours de développement pour le M-Calcul, en particulier la conception d'une machine abstraite, ainsi que la définition d'observables afin de définir des équivalences entre processus.

Ce travail sera présenté à la conférence POPL 2003[17], Janvier 2003, à la Nouvelle-Orléans.

6.5. Typage des modules mobiles

Participants : Georges Gonthier, James Leifer, Jean-Jacques Lévy, Gilles Peskine.

Il s'agit de concevoir un typage sûr de l'emballage (*marshalling*) des données dans un environnement distribué, avec une notion d'égalité de type utilisable entre les différents composants d'un système distribué. Le principe est de donner suffisamment d'information pour qu'une vérification de type puisse être faite lors du déballage d'une donnée transmise pour vérifier l'adéquation du type de la valeur envoyée au type attendu. Ainsi il est possible de détecter les erreurs de type aussi tôt que possible, c'est-à-dire à la compilation ou au moment du déballage, plutôt qu'au moment de l'utilisation de la donnée.

Une des caractéristiques principales de ML est la notion de module avec la possibilité de cacher la représentation d'un type abstrait. Par exemple, un programmeur utilise une représentation des ensembles par des listes triées. La logique du programme repose sur cette propriété dont il sait qu'aucune fonction externe au module ne pourra mettre en défaut. Si un système d'emballage ne fait que stocker le type de la représentation concrète de la valeur emballée, on ne pourra maintenir l'invariance de telle propriété, puisqu'alors une liste non triée sera considérée comme valide dans l'exemple précédent.

Nous proposons un calcul avec un typage reliant les types de données abstraits par une fonction de hachage sur les constructeurs de ce type. Ce calcul permet l'égalité structurée de type et est la base pour une étude des modules paramétriques. Ce calcul repose sur un lambda calcul étendu par des modules, inspiré des systèmes de Harper et Lillibridge, et avec des primitives pour l'emballage, le déballage, et la communication. Une règle de réduction transforme les types abstraits en signatures du module qui les implémente. Ainsi, les propriétes abstraites de la représentation sont préservées. On peut donc lire les valeurs abstraites générées par un module,

éventuellement sur une machine distante, et les manipuler en local, en ne vérifiant que les informations de type signées sont valides au moment du déballage.

Ce travail a été démarré par une coopération avec P. Sewell et K. Wansbrough de Cambridge ; il est toujours en cours. La préservation du type et le théorème de progression ont été démontrés. G. Peskine and J. Leifer ont également réalisé un prototype sur une pré-version du typage en modifiant le compilateur Ocaml.

D'autre part G. Gonthier a repris ses travaux avec M. Abadi (UC Santa Cruz) et B. Werner (LogiCal) sur l'utilisation du symbole de choix de Hilbert, ε [24], pour modéliser le typage de l'édition de lien dynamique (*dynamic linking*) de composants logiciels. Il sont parvenus à définir un fragment du système de types dans lequel les problèmes théoriques et pratiques qu'il pose (notamment la gestion des versions) admettent une solution raisonnable. Ces travaux seront soumis à publication en 2003.

6.6. Analyse de flot dans le join-calcul

Participant : Sylvain Conchon.

Les problèmes de sécurité peuvent se voir comme un problème de contrôle de flot dans l'accès à des canaux. Un typage (statique) faisant intervenir divers degrés de confidentialité à un canal a été développé. Le travail a considéré les cas du lambda-calcul, du join-calcul et du pi-calcul. Cette analyse de contrôle de flot dans le join-calcul a été conclu cette année par la soutenance de thèse de S. Conchon [7], parti à OGI en janvier 2002.

6.7. Extension objet du join-calcul

Participants : Luc Maranget, Ma Qin.

La conception d'une extension objet du Join-Calcul est un travail de longue haleine. Un travail initial mené en collaboration avec D .Rémy du projet Cristal, C. Fournet (Microsoft Research) et C. Laneve (Université de Bologne) a donné lieu à une publication à la conférence *Foundations of Software Technology and Theoretical Computer Science* en 2000. La version longue est soumise à publication.

Le travail de thèse de Ma Qin consiste à étudier comment intégrer une telle extension dans un compilateur du join-calcul. L'article propose en effet un système que l'on peut penser exagéremment compliqué pour la programmation. On peut envisager de le rendre moins expressif pour un gain important en simplicité. Par ailleurs, l'inférence des types de classe est possible, La recherche de Ma Qin consiste à trouver une implémentation utilisable en pratique de cette inférence. Un autre objectif significatif est la recherche d'une réalisation efficace de l'appel de méthode, en combinaison avec le filtrage des motifs du join-calcul. Dans ces deux dernier domaines, des arbitrage entre complexité, tant au sens des algorithmes qu'à celui des concepts, et expressivité sont probables.

6.8. Analyseur de code objet pour Caml-light

Participant : Sébastien Ailleret.

Le prototype d'un analyseur statique de code objet pour Caml-light a été développé en utilisant une méthode du style *proof carrying code*. Il a fait l'objet d'une présentation à JFLA'02, Chamrousse[12]. S. Ailleret travaille à PSA depuis la fin janvier 2002.

6.9. Implantation d'une démonstration du théorème des quatre couleurs dans le calcul des constructions

Participant : Georges Gonthier.

Le théorème des quatre couleurs affirme qu'il est possible de colorier avec seulement quatre couleurs les régions de toute carte planaire, tout en donnant des couleurs différentes à toute paire de régions adjacentes. C'est l'un des résultats les plus célèbres de mathématiques discrètes, célébrité due en grande partie au fait qu'il ait fallu un siècle et l'utilisation des ordinateurs pour le démontrer. Cependant, cette utilisation des machines a

toujours laissé susbsister un doute sur sa validité en particulier à cause du passage par la sémantique informelle du langage machine qu'elle implique.

Ces caractéristiques font du théorème des quatre couleurs un exemple idéal pour éprouver et démontrer la maturité d'un système de démonstration automatique comme le système Coq du projet LogiCal. Fort de son expérience en 1995 de la vérification d'un algorithme de récupération de mémoire concurrente, G. Gonthier a entrepris, avec B. Werner (LogiCal) et V. Danos (Paris VII), l'implantation en Coq d'une preuve récente (1995) du théorème des quatre couleurs due à Robertson, Sanders, Seymour et Thomas [29].

Comme la preuve d'Appel et Haken de 1974 [23], cette preuve se décompose en deux parties :

1. vérifier, par énumération des sous-cartes de rayon 2, composées régions penta- à octagonales, que la formule d'Euler implique que toute carte planaire contient l'une des sous-cartes d'une liste de 633 sous-cartes particulières,
2. vérifier, par analyse de l'ensemble de ses coloriages, que chacune de ces 633 sous-cartes est *réductible*, c'est-à-dire qu'elle permet de ramener le coloriage de toute carte qui la contient à celui d'une carte plus petite.

C'est en fait la seconde partie qui a exigé l'emploi de l'ordinateur, historiquement ; Appel et Haken avaient étudié manuellement quelques 10 000 cas de figure pour établir le 6.9, mais cette bravoure n'eût pas été possible pour les centaines de millions de cas du 6.9.

C'est donc par le 6.9 que nous avons commencé. Pour cette démonstration nous nous sommes appuyés sur la possibilité offerte par le calcul inductif des constructions de réfléchir un calcul combinatoire dans le calcul de types, ce qui nous a permis d'écrire un programme dans la preuve, et de limiter la taille du témoin de preuve que le système Coq produit. Bien que nous ayons été contraints d'utiliser un algorithme d'énumération un peu sophistiqué, afin de pallier aux contraintes de performances de la « machine » de typage de Coq (notamment, l'absence de tableaux), cette partie n'a pas posé de difficultés. Un des sous-produits de cet effort a été le développement d'un style de script de preuve permettant de combiner efficacement étapes déductives, équationelles et calculatoires ; plusieurs traits de ce style ont déja été intégrés dans les dernières diatributions de Coq.

Nous avons en revanche rencontré des difficultés dans l'implantation de la métathéorie reliant 6.9 et 6.9, bien qu'il ne s'agisse de prime abord que de résultats assez simples de théorie des graphes. En effet les présupposés topologiques sur lesquelles se fondent les démonstrations « intuitives » de ces résultats ne sont pas accessibles depuis Coq, sauf à refaire la théorie de l'intégration complexe pour étalir le théorème de Jordan. Nous avons donc été conduits à poser une nouvelle définition, purement combinatoire, des graphes planaires. Cette définition est hautement symétrique, puisqu'elle étend la dualité face-sommet habituelle à une triple dualité face-arête-sommet, et elle met en lumière le rôle central de la formule d'Euler. Cette nouvelle approche nous a permis de retrouver les résultats de base, et en particulier une forme purement combinatoire de la formule de Jordan, dont la démonstration constitue une première dans le domaine des systèmes formels.

6.10. Logiques liantes

Participant : Thérèse Hardin.

En collaboration avec G. Dowek (projet Logical) et C. Kirchner (projet Prothéo), T. Hardin a continué ses travaux sur les modèles de la logique avec des symboles lieurs. Ils ont en particulier proposé un contre-modèle pour l'axiome d'extensionalité. Ce travail a été publié à LPAR02 [14].

6.11. Projet Foc

Participants : Thérèse Hardin, Damien Doligez, Virgile Prévosto.

T. Hardin dirige le projet de développement d'un environnement de programmation certifiée pour le Calcul formel, appelé FOC. Les participants de ce projet sont S. Boulmé, M. Jaume, S. Fechter, V. Ménissier-Morain,

V. Prévosto et R. Rioboo, membres du LIP6, D. Doligez du projet Moscova, D. Delahaye, C. Dubois, O. Pons et V. Viguié Donzeau-Gouge du CEDRIC (CNAM).

Le projet FOC développe un langage dédié au calcul formel, permettant à ses utilisateurs de déclarer et définir des fonctions, d'énoncer des propriétés et d'en faire la preuve. La librairie du langage est structurée en *espèces*.

Les espèces permettent d'abord de spécifier une structure algébrique, exactement comme en mathématiques. Par exemple, spécifier un semi-groupe unitaire consiste à définir une espèce *semi-group* c'est-à-dire à nommer l'ensemble sous-jacent, l'opération sur cette structure (la multiplication) et à nommer un élément distingué, l'unité. Les axiomes du semi-groupe sont exprimés par des propriétés et leurs conséquences - des théorèmes - peuvent être démontrées. On peut ensuite utiliser les espèces de différentes manières pour construire d'autres espèces.

- Par exemple, on peut construire une espèce représentant les groupes en héritant de *semi-group*. Il suffit alors d'ajouter l'opération d'inverse et quelques propriétés. De même, les anneaux sont construits par *héritage multiple* du groupe additif et du monoïde multiplicatif.
- On peut aussi, à partir de *semi-group*, construire un semi-groupe ayant comme support un ensemble fini, par exemple, en précisant l'ensemble sous-jacent et en définissant la table de multiplication correspondante. Il faudra alors démontrer que les propriétés caractéristiques des semi-groupes sont satisfaites par cette implantation. Cette opération, qui spécialise une espèce en apportant des précisions et des définitions, est appelée *raffinement*.
- Lorsque les structures s'enrichissent, certaines définitions peuvent devenir obsolètes et doivent être remplacées par des définitions tirant profit des nouvelles propriétés. Ce mécanisme est bien connu dans la programmation orientée-objet, où il est appelé *liaison retardée*.
- Les espèces peuvent être paramétrées par des types (celui de l'ensemble-support par exemple) mais aussi par des valeurs et d'autres espèces.
- Enfin, une espèce peut être appauvrie par oubli d'opérations et/ou de propriétés, par exemple, pour être passée en paramètre effectif à une autre espèce. Ainsi, l'anneau des polynômes à une variable est paramétrée par l'espèce spécifiant l'anneau des coefficients. On peut bien sûr choisir un corps comme anneau de coefficients.

Les *collections* sont construites à partir d'espèces complètement définies, par abstraction de la représentation de l'ensemble sous-jacent à la structure.

La famille des espèces et des collections constitue donc un graphe, dont les arcs correspondent à la composition des opérations mentionnées ci-dessus. Dans la mesure où la définition d'une espèce contient des énoncés de propriétés et des preuves, il est important de s'assurer de la cohérence de l'ensemble de cette hiérarchie, ce qui a été fait par S. Boulmé. La notion d'espèce a été spécifiée en Coq, ainsi que les différents moyens d'étendre la hiérarchie des espèces. Des critères permettant de garantir la cohérence (en particulier, en cas d'héritage multiple et de redéfinition) ont été dégagés. Afin de s'assurer que l'utilisation de Coq ne conduisait pas à une sur-spécification, un modèle catégorique de la hiérarchie a été construit. Ce modèle n'utilise pas de syntaxe concrète pour décrire la hiérachie mais uniquement des propriétés requises par l'analyse.

Côté programmation, la bibliothèque doit fournir des programmes non seulement corrects mais aussi efficaces. L'efficacité est à prendre en compte à différents niveaux : expressivité, distance réduite entre le source et sa spécification de manière à faciliter les preuves, portabilité sur différentes architectures, efficacité à l'exécution (en temps et en place mémoire). Un certain nombre de prototypes (d'une cinquantaine de classes chacun) ont été développés en Ocaml (par R. Rioboo), afin de de déterminer la discipline d'utilisation des traits objet et modules de Ocaml. Celle qui a été adoptée minimise la distance entre la spécification de la hiérarchie et des espèces en Coq et leurs implantations en OCaml.

Le développement du langage de FOC est effectué par D. Doligez et V. Prévosto. A partir de la spécification en Coq et des études déterminant la cible pour la programmation, ils ont défini une syntaxe concrète, très

expressive, qui permet de regrouper dans un même texte des déclarations, définitions, énoncés et preuves. Ce texte sert de base à un double processus de traduction vers Ocaml et vers Coq.

La syntaxe interdit tout écart à la discipline de programmation et les outils offerts pour la création de nouvelles espèces implantent les spécifications des mécanismes d'extension spécifiés dans Coq. Ces propriétés du source FOC sont garanties d'abord par une synthèse/vérification des types et des mécanismes d'importation. Ensuite, les déclarations, définitions, énoncés et preuves introduisent de multiples dépendances entre identificateurs. La préservation de la cohérence de la hiérarchie est assurée par une analyse statique de ces dépendances, fondée sur la spécification en Coq. Cette analyse a été présentée à la conférence TPHOL2002[13]. Une fois analysé, le source FOC est d'une part compilé vers un source OCaml, qui respecte la discipline de programmation. L'efficacité du code ainsi obtenu est excellente.

Le source FOC est d'autre part compilé vers Coq comme suit. Il n'a pas été possible de réutiliser les développements effectués par S. Boulmé pour des raisons d'efficacité (termes de taille trop importante). A partir de l'analyse statique du texte source, le compilateur de preuves construit un contexte d'hypothèses en utilisant les déclarations d'héritage, de raffinement, d'instanciation de paramètres, etc. L'énoncé de la propriété, qui dans la syntaxe Foc ressemble fort à une formule du premier ordre, est placé dans ce contexte. Sa preuve est effectuée de manière interactive, soit en utilisant un ensemble (encore restreint) de tactiques propres, soit en utilisant celles de Coq. Le processus de preuve se termine par la construction d'un terme preuve soumis à Coq pour vérification.

Alors que la compilation vers OCaml peut être considérée comme achevée, la construction de la preuve demande encore un certain nombre de développements, tant au point de vue théorique qu'au point de vue pratique. Sur le plan théorique, D. Doligez et T. Hardin étudient les propriétés d'une extension de la logique du premier ordre introduite par D. Doligez, utilisée pour effectuer la preuve d'un énoncé sous hypothèses. Avec V. Prévosto, ils étudient également certaines restrictions des langages de types dépendants, pouvant suffire à construire une preuve complète. Il reste de nombreuses extensions à effectuer sur le plan pratique, en particulier le traitement de l'induction.

En l'état actuel, la librairie fournit les algorithmes standard en algèbre commutative, des arithmétiques entières (entiers machine, entiers modulaires, entiers en précision arbitraire en Ocaml, grands entiers offerts par GMP3) et rationnelles. Elle propose également une arithmétique générique des polynômes sur un anneau commutatif quelconque, avec toutes les représentations habituelles (denses, creuses, récursives). L'alogorithme de Loos pour le calcul des résultants de polynômes est aussi disponible ainsi que la factorisation des polynômes à une variable sur un corps fini (algorithme de Cantor-Zassenhaus revu par Quitte-Naudin). Les extensions algébriques de corps sont aussi disponibles. Il faut cependant noter que nombre de preuves restent à faire pour ces différentes espèces et collections. Leur complexité justifie les efforts faits par D. Doligez et V. Prévosto pour assurer une bonne ergonomie de FOC. Une première distribution du langage est prévue au printemps 2003.

6.12. Énumérateurs fonctionnels

Participant : Luc Maranget.

Luc Maranget a développé une technique de décision simple s'appliquant à la logique des prédicats monadiques. La technique permet la production d'énumérateurs à partir de la définition dénotationnelle des opérateurs logiques. A notre avis, cette technique est simple, élégante et inédite. Elle s'apparente à l'énumération systématique de toutes les valeurs des variables sur un domaine fini avec un système de continuations. Toutefois le domaine n'est pas parcouru complètement, de sorte que cette technique d'énumeration se montre utilisable en pratique, là où la technique d'énumeration naïve est impraticable. Ce travail est à comprendre comme un exemple de « belle » programmation fonctionnelle. Il a eté accepté pour publication dans le *Journal of Functionnal Programming* [11].

7. Contrats industriels

7.1. Méthodologie pour la robustesse des logiciels spatiaux

Participants : Georges Gonthier, Jean-Jacques Lévy.

Avec la finalisation de l'étude de l'état de l'art au printemps 2002, et son incorporation dans une proposition de méthodologie par Terma A/S au cours de l'été 2002, la collaboration avec la société danoise Terma sur une étude sur une méthodologie pour la robustesse des logiciels spatiaux s'est achevée en 2002. La méthodologie devrait être déployée par la suite dans des projets industriels par Terma.

7.2. CoordinAtion et Répartition des Applications Multiprocesseurs en OCaml

Participants : Georges Gonthier, Gilles Peskine.

En 2002, les efforts dans cette Action Concertée Initiative 2001, dirigée par Gaëtan Hains (U. Orléans), se sont concentrées sur le développement de bibliothèques pour la parallélisation locale des calculs haute-performance en OCaml. Le passage au calcul distribué, prévu pour 2003, verra une plus grande implication de Moscova.

7.3. Ozone

Participant : Georges Gonthier.

Le projet Moscova participe, sous la coordination de V. Issarny (Arles) à l'important effort de l'Inria dans le projet Européen Ozone, avec Philipps et Thomson Multimédia, sur la réalisation d'une plateforme de démonstration pour l'intelligence ambiante.

2002 était l'année de démarrage du projet, et de la définition de architecture de la plateforme. L'idée d'une architecture agent, basée sur JoCaml, a dû être écartée au profit d'une architecture client-serveur plus classique, et les efforts de Moscova se sont concentrés sur la conception de l'architecture de sécurité, qui devraient être sa principale contribution au projet pour 2003-2004.

7.4. Pepito

2002 était la première année du projet PEPITO (*Peer to Peer, Implementation and Theory*), accepté dans l'appel d'offre *Global Computing* de Esprit. Ce projet regroupe des équipes de KTH (Seif Haridi, coordinateur), Cambridge (Peter Sewell), EPFL (Martin Odersky), SICS (Per Brand), UCL (Peter van Roy). Ce projet traite des aspects théoriques et pratiques des applications symétriques et distribuées, avec mobilité. Il fédère les efforts déjà développés autour de Mozart (Mobile Oz), JoCaml et Funnel (Functional Nets).

La première réunion scientifique de Pepito a eu lieu en janvier 2002 à Rocquencourt. Il y a eu entre autres des exposés sur un prototype de détecteur de pannes distribué en Ocaml (J. Leifer), sur la sémantique des réseaux UDP (P. Sewell, Cambridge), sur la modélisation du consensus distribué dans les calculs de processus (U. Nestmann, Lausanne), et sur la programmation dans le système Oz (P. van Roy, Louvain).

La seconde réunion a eu lieu et juillet 2002 à Stockholm, avec des exposés sur le M-calcul (A. Schitt), sur le typage de la transmission de données (« type-safe marshalling », J. Leifer), sur les algorithmes distribués « indulgents » (R. Guerraoui, Lausanne), et sur le stockage de données dans les réseaux peer-to-peer (Per Brand, SICS).

Le travail sur le M-calcul et la liaison dynamique (Schmitt), et sur les algorithmes distribués pour la transmission sûre et typée de types abstraits de données (Leifer, Lévy, Peskine) constituent les principales contributions de Moscova pour la première année de Pepito [18].

7.5. France Telecom

J.-J. Lévy co-supervise, avec P. Crégut, la thèse de Guillaume Chatelet, inscrit à l'Ecole polytechnique, travaillant à France Telecom sur les extensions du langage Erlang avec des primitives de migration.

8. Actions régionales, nationales et internationales

8.1. Accueils de chercheurs étrangers

Nos visiteurs en 2002 ont été Carlos Varela, Rensselaer, (15 juillet) P. Sewell, Cambridge, (9-13 septembre), Leslie Lamport, Microsoft Research, (4 décembre). Vladimiro Sassone, Univ. de Sussex (30 septembre), Cosimo Laneve, Univ. de Bologne (septembre). David MacQueen, Univ. de Chicago, est resté 4 mois (septembre - décembre) dans notre projet, à temps partiel.

9. Diffusion des résultats

9.1. Animation de la communauté scientifique

G. Gonthier est vice-président du comité des projets de l'INRIA Rocquencourt, et membre suppléant de la commission d'évaluation depuis septembre 2002. Il a participé au jury CR de Rocquencourt en mai 2002. Il représente l'INRIA au comité directeur des Écoles d'été CEA-EDF-INRIA.

T. Hardin est vice-présidente de SPECIF, chargée de la recherche, depuis juin 2001.

D. Doligez a participé au jury de thèse de Teresa Higuiera-Toledano (Rennes).

T. Hardin a été membre des jurys de thèse de Pascal Cuoq (INRIA-LIP6), Grégoire Hamon (LIP6), Mirna Bognar (Vrije U., Amsterdam), rapporteur des thèses de Bruno Barbier (Besaçon), Eric Deplagne (Nancy IHP), directrice de l'habilitation de Renaud Rioboo (LIP6) et rapporteur de l'habilitation de Chouki Aktouf (Valence-LCIS).

J.-J. Lévy a été membre des jurys de thèse de Cédric Lousshaine (Marseille), S. Conchon (Paris 7) et A. Schmitt (Polytechnique), et de l'habilitation de D. Sangiorgi (Paris 7), de M. Shapiro (Paris 6) et de M. Pouzet (Paris 6).

9.2. Enseignement universitaire

Notre projet participe aux enseignements suivants :

- Informatique Fondamentale, cours de deuxième année à l'École polytechnique, 250 élèves, (J.-J. Lévy, professeur responsable du cours, G. Gonthier, professeur chargé de cours, A. Schmitt et J. Leifer vacataires) ; G. Gonthier a été responsable des projets informatique ; J.-J. Lévy a produit un nouveau polycopié à l'automne 2002 (270 pages), complètement disponible sur le Web [21].
- Compilation, cours de troisième année à l'École polytechnique, 20 élèves, (L. Maranget, professeur chargé de cours, responsable du cours ; A. Schmitt, vacataire y a assuré les travaux pratiques). Un polycopié est disponible sur le Web [22].
- Lambda calcul, cours du DEA DEA Sémantique, preuves et programmation. (T. Hardin, et petite participation de J.-J. Lévy).
- Concurrence et mobilité, cours du DEA DEA Sémantique, preuves et programmation. (G. Gonthier).
- Développement des logiciels sûrs, DESS DLS, CNAM, Paris 6 et ENS Cachan. (T. Hardin, professeur, co-responsable.

Moscova est équipe d'accueil de l'école doctorale EDITE.

L. Maranget est l'auteur du problème d'informatique de l'épreuve (4h) de l'option MP-informatique au concours d'entrée à l'École polytechnique en 2002. J.-J. Lévy a participé à l'élaboration des sujets de la nouvelle épreuve (2h) d'informatique pour tous du concours d'entreé à l'École polytechnique en 2002 pour les candidats n'ayant pas choisi l'option MP-informatique.

9.3. Participation à des colloques, séminaires, invitations

9.3.1. Participation à des congrès

- 1-2 février : F. le Fessant, G. Gonthier, J. Leifer, J.-J. Lévy, L. Maranget, Ma Qin, G. Peskine, A. Schmitt, au premier workshop du projet PEPITO, Rocquencourt. J. Leifer a donné un exposé sur « Implementations of failure detection à la Toueg ».
- 29 juin - 3 juillet : J.-J. Lévy, J. Leifer, A. Schmitt ont participé au deuxième workshop du projet PEPITO, Stockholm. J. Leifer a donné un exposé sur « Safe marshalling in Jocaml : the design space ». A. Schmitt a donné un exposé sur le M-calcul.
- 7 septembre : G. Gonthier a participé, avec les chercheurs de Thomson et Philips, à la réunion du projet européen Ozone sur la sécurité.
- J. Leifer a fait partie du comité de programme d'Express 2002 (mené par P. Panangaden et U. Nestmann).
- août : A. Schmitt a présenté un article sur le join-calcul dynamique à la conférence IFIP-TCS, Montréal.
- juillet : T. Hardin a participé au symposium Calculemus 2002, Marseille.
- 10-14 avril : D. Doligez, T. Hardin, et V. Prévosto ont assisté au workshop *35 years of Automath*, Edimbourg.
- 19-23 mai : D. Doligez a assisté au workshop sur le calcul formel libre, Lyon.
- 19-21 juin : D. Doligez, T. Hardin ont assisté au groupe de travail FOC, Chamrousse.
- 12-15 novembre : L. Maranget a assisté à l'atelier scientifique pluridisciplinaire *Sexes et genres dans le travail scientifique* à Cargèse.

9.3.2. Visites

- 14-19 mars : J. Leifer et G. Peskine ont rendu visite au Computer Laboratory, Cambridge, pour travailler avec P. Sewell sur la liaision dynamique dans les programmes distribués, et avec R. Milner sur les bisimulations congruentes pour systèmes « bigraphes ».
- 6-7 avril : J. Leifer a rendu visite à U.Nestmann à l'EPFL, Lausanne pour travailler sur la modélisation du consensus avec les algèbres de processus.
- 13 septembre - 6 octobre : J. Leifer a rendu visite au Computer Laboratory, Cambridge, pour travailler avec P. Sewell sur le typage de l'emballage de données (*marshalling*).
- 9-13 décembre : G. Gonthier s'est rendu à l'Université de Californie à Santa Cruz, pour reprendre ses travaux avec M. Abadi et B. Werner (LogiCal) sur l'utilisation du epsilon-calcul de Hilbert pour le typage de l'édition de lien dynamique de composants logiciels.

9.3.3. Exposés

- A. Schmitt a présenté le M-Calcul au séminaire PPS de Paris 7.
- T. Hardin a donné une conférence invitée aux JFLA2002 sur « Les logiciels de confiance ».
- D. Doligez et T. Hardin ont donné une conférence invitée au workshop « Thirty five Years of Automath », Edimbourg, avril 2002, sur le système FOC.

10. Bibliographie

Bibliographie de référence

[1] M. ABADI, L. CARDELLI, P.-L. CURIEN, J.-J. LÉVY. *Explicit Substitutions.* in « Journal of Functional Programming », numéro 4, volume 1, 1991, pages 375-416.

[2] M. ABADI, C. FOURNET, G. GONTHIER. *Secure implementation of channel abstractions.* in « Proceedings of the Thirteenth Annual IEEE Symposium on Logic in Computer Science », pages 105-116, juin, 1998.

[3] P.-L. CURIEN, T. HARDIN, J.-J. LÉVY. *Confluence Properties of Weak and Strong Calculi of Explicit Substitutions.* in « Journal of the ACM », numéro 2, volume 43, mars, 1996, pages 362-397, ftp://theory.lcs.mit.edu/pub/jacm/jacm.bib.

[4] A. DEUTSCH. *On The Complexity of Escape Analysis.* in « 24th Annual ACM Symp. on Principles of Programming Languages », ACM Press, pages 358-371, janvier, 1997.

[5] C. FOURNET, G. GONTHIER. *The Reflexive Chemical Abstract Machine and the Join-Calculus.* in « Proceedings of the 23rd Annual Symposium on Principles of Programming Languages (POPL) (St. Petersburg Beach, Florida) », ACM, pages 372-385, janvier, 1996.

[6] C. FOURNET, G. GONTHIER, J.-J. LÉVY, L. MARANGET, D. RÉMY. *A Calculus of Mobile Agents.* in « CONCUR '96 : Concurrency Theory (7th International Conference, Pisa, Italy, August 1996) », série LNCS, volume 1119, Springer, éditeurs U. MONTANARI, V. SASSONE., pages 406-421, 1996.

Thèses et habilitations à diriger des recherche

[7] S. CONCHON. *Analyse modulaire de flot d'information pour les calculs séquentiels et concurrents.* thèse de doctorat, Paris 7, Septembre, 2002.

[8] A. SCHMITT. *Conception et Implémentation de Calculs d'Agents Mobiles.* thèse de doctorat, Ecole polytechnique, Septembre, 2002.

Articles et chapitres de livre

[9] M. ABADI, C. FOURNET, G. GONTHIER. *Secure Implementation of Channel Abstractions.* in « Information and Computation », numéro 174, 2002, pages 37-83.

[10] C. FOURNET, G. GONTHIER. *The Join Calculus : A Language for Distributed Mobile Programming.* éditeurs L. P. G. BARTHE, J. SARAIVA., in « Applied Semantics », série Lecture Notes in Computer Science, volume 2395, Springer Verlag, mai, 2002, pages 268-332.

[11] L. MARANGET. *Functional Satisfaction.* in « Journal of Functionnal Programming », à paraître en 2003.

Communications à des congrès, colloques, etc.

[12] S. AILLERET. *Typage du bytecode Caml.* in « Actes des journées francophones des langages applications », janvier, 2002.

[13] D. DOLIGEZ, T. HARDIN, V. PRÉVOSTO. *Algebraic Structures and Dependent Records.* in « Proceedings of TPHOL'03 », éditeurs B. RICK., NASA, Hampton, USA, 2002.

[14] G. DOWEK, T. HARDIN, C. KIRCHNER. *Binding logic : proofs and models.* in « Logic for Programming, Artificial Intelligence, and Reasoning », volume 2514, Springer-Verlag, éditeurs M. BAAZ, A. VORONKIV.,

pages 130-144, 2002.

[15] L. MARANGET. *Les avertissements du filtrage.* in « Actes des journées francophones des langages applications », janvier, 2002.

[16] A. SCHMITT. *Safe Dynamic Binding in the Join Calculus.* in « Proceedings of IFIP/TCS », Montreal, 2002.

[17] A. SCHMITT. *The M-calculus : a Higher-Order Distributed Process Calculus.* in « Proceedings of ACM/POPL Conf. », New Orleans, janvier, 2003.

Rapports de recherche et publications internes

[18] S. HARIDI, T. SJOLAND, SICS, KTH, UCAM, INRIA, UCL, EPFL. *PEPITO (PEer-to-Peer Implementation and TheOry) Periodic Progress Report Number 1.* rapport technique, janvier, 2003, Project funded by the European Community under the Information Society Technologies Programme (1998-2002).

[19] J. LEIFER. *Synthesising Labelled Transitions and Operational Congruences in Reactive Systems, Part 1.* rapport technique, numéro 4394, INRIA, Rocquencourt, mars, 2002, http://www.inria.fr/rrrt/rr-4394.html, soumis à MSCS.

[20] J. LEIFER. *Synthesising Labelled Transitions and Operational Congruences in Reactive Systems, Part 2.* rapport technique, numéro 4395, INRIA, Rocquencourt, mars, 2002, http://www.inria.fr/rrrt/rr-4395.html, soumis à MSCS.

[21] J.-J. LÉVY. *Informatique Fondamentale.* Ecole polytechnique, Décembre, 2002.

[22] L. MARANGET. *Cours de compilation.* Ecole polytechnique, Décembre, 2002, http://www.enseignement.polytechnique.fr/profs/inform

Bibliographie générale

[23] K. APPEL, W. HAKEN. *Every planar map is four colourable.* in « Contemporary Mathematics », volume 98, 1989, pages 1-741.

[24] J. AVIGAD, R. ZACH. *The epsilon calculus.* in « the Stanford Encyclopedia of Philosophy », mai, 2002, http://plato.stanford.edu/entries/epsilon-calculus.

[25] C. A. R. HOARE. *Communicating Sequential Processes.* Prentice Hall, 1985.

[26] R. MILNER. *Communication and Concurrency.* Prentice Hall, 1989.

[27] R. MILNER. *Calculi for interaction.* in « Acta Informatica », numéro 8, volume 33, 1996, pages 707-737.

[28] R. MILNER, J. PARROW, D. WALKER. *A Calculus of Mobile Processes, Parts I and II.* in « Journal of Information and Computation », volume 100, septembre, 1992, pages 1-77.

[29] P. S. N. ROBERTSON, R. THOMAS. *The Four Color Theorem.* in « Journal of Combinatorial Theory », volume B 70, 1997, pages 2-44.

[30] B. C. PIERCE, D. N. TURNER. *Pict : User Manual.* 1997, Available electronically.

[31] B. THOMSEN, L. LETH, T.-M. KUO. *A Facile Tutorial.* in « CONCUR '96 : Concurrency Theory (7th International Conference, Pisa, Italy, August 1996) », série LNCS, volume 1119, Springer, éditeurs U. MONTANARI, V. SASSONE., pages 278-298, 1996.

[32] ROBIN MILNER. *The Polyadic π-Calculus : a Tutorial.* rapport technique, numéro ECS-LFCS-91-180, LFCS, University of Edinburgh, octobre, 1991, Also in *Logic and Algebra of Specification*, ed. F. L. Bauer, W. Brauer and H. Schwichtenberg, Springer, 1993.