Activités d' enseignement et de recherche de Guillaume Gillard

Certilab - INRIA Sophia Antipolis
2004 route des luciole B.P 93
06902 Sophia Antipolis Cedex
`Guillaume.Gillard@sophia.inria.fr`

1 Présentation Générale

Je suis actuellement doctorant de l' université Denis Diderot (Paris 7) rattaché au laboratoire de recherche de l' INRIA Sophia Antipolis. Mes codirecteurs de thèse sont Joëlle Despeyroux (INRIA) et Guy Cousineau (Paris 7). Ma thèse, en cours de rédaction, a pour titre "**Sémantique en COQ: vers la spécification et les preuves sur des langages à objets**".

Contexte scientifique.
Langages orientés objets: Le développement des langage à orientés objets connaît un fort succès dans les domaines de la programmation tant académique qu' industrielle. La programmation à grande échelle nécessite une méthodologie et un support qui permettent au moins la compilation séparée la protection des détails d'implémentation et la réutilisation du code. Pour obtenir cela, on retrouve dans différents langages des mécanismes tels que la compilation séparée, types de données abstraits et l'héritage. Les difficultés, la grande complexité mais aussi la richesse de la plupart des langages dits à objets viennent de l'ambition de ces langages de capturer tous ces aspects à la fois, et cela en utilisant une seule notion, celle d'objet.

Langages concurrents: Le partage de l' information et des taches à réaliser entre plusieurs machines a initier un réel besoins de modélisation des systèmes réactifs, parallèles et communicants. La classe de modèles, que j' ai choisi d' adoptée est celle des calculs de processus,formalismes algébriques simples pour la représentation des systèmes distribués et des processus concurrents. Des langages comme *Pict* sont directement issus de ces recherches. Les principaux domaines d'application visés couvrent les protocoles et les systèmes et algorithmes répartis. Une autre classe de modèles importante est celle des formalismes réactifs synchrones, dont font partie les langages Esterel et RC. Ce type de programmation est particulièrement adapté aux systèmes temps réel,manipulant des événements logiques avec préemption.

Sécurité et formalisation: La fiabilité et la robustesse des programmes, sont devenues aujourd'hui de véritables défies tant académique qu' industrielle. Les domaines où la présence d'erreurs même minimes dans les programmes peut avoir des conséquences graves : danger pour la vie humaine comme dans l'énergie nucléaire ou les transports (avions, automobiles), la chirurgie assistée par ordinateur, ou simplement coût prohibitif comme dans le cas du commerce électronique et des réseaux de télécommunication, ou du spatial ont de réels besoins de programmes dit zéro défauts. Parmi les différentes méthodes qui permettent d' obtenir des programmes sures celle de la preuve assistée par ordinateurs connaît un développement sans précédent depuis quelques années. Actuellement, les champs d'application de ces méthodes restent limités notamment pour des raisons de faisabilités. Parmi tout les domaines potentiel d' application telles méthodes celui de l' étude *formelle* des langages de programmation semble particulièrement prometteur. D' une part les exigences en matière de fiabilité sont forte, de l' autre, ces technique semblent en mesure de répondre aux attentes des différents acteurs du domaine.

Stage de DEA.
J' ai effectué mon stage de DEA au sein de projet CROAP à l ' INRIA Sophia Antipolis sous la direction de Joëlle Despeyroux. L' objectif était notamment, de spécifier formellement un *mini-ML*, une machine à

pille (*CAM*) leur sémantique, un compilateur (mini_ML-CAM) et de fournir une preuve en *Coq* de correction du compilateur au regard des sémantique. J' ai acquis une première connaissance de l' assistant de démonstration *Coq* ainsi que de l' interface graphique *CtCoq* développe dans le projet.

2 Sujet de recherche et thèse

Outils d' aide à la preuve

Un outil d' aide a la preuve est un système de manipulation de preuves mathématiques; une preuve réalisée par un tel système est mécaniquement vérifiée par la machine. Parmi les différents outils disponible actuellement, j' ai choisi d' utiliser le système *Coq* développer à l' INRIA. Ce système permet de manipuler des assertions du calcul, de vérifier mécaniquement des preuves de ces assertions, d'aider à la recherche de preuves formelles et de synthétiser des programmes certifiés à partir de preuves *constructives* de leurs spécifications.
On appel une preuve ou un raisonnement *formel*, toute preuve qui utilise un tel outils. Par opposition, un raisonnement ou une preuve informel (aussi rigoureuse soit elle) est une preuve qui n' en utilise pas; comme une preuve en mathématique *sur papier*. Une spécification formelle est la donnée dans le langage de spécification de ces outils d' une spécification informelle.

Formalisations et α-conversion

Lors de raisonnements informels sur des calculs on utilise la plus part du temps non pas l' égalité sur les termes mais la relation dite d' α-équivalence entre eux. Cette relation exprime le faite que le nom d' une variable *liée* n' est pas significatif et donc interchangeable. Il arrive, que cette relation soit étudier en détails (J.L.Krivine Lambda-calcul types et modèles) mais, bien souvent, on admet simplement (à raison) que cette relation a les *bonne propriétés*. Par abus de langage les termes du calculs sont identifiés avec leur classe d' équivalence (en mathématiques on dira que la fonction $x \mapsto x$ est *égale* à la fonction $y \mapsto y$). Ainsi, il suffit de supposer que l' on raisonne toujours avec un représentant adéquat de la classe d' équivalence.

Cette méthode de raisonnement n' est pas viable lors de raisonnements formels. Avec de tels outils, impossible d' exhiber avec une telle facilite un représentant adéquat qui risque de changer au cours du raisonnement.

On retrouve le lieur du λ-calcul dans les calculs à objets principalement sous le forme du *self* (ou *this* en java) qui *lie* l' objet et ses méthodes. Les calcules concurrents possèdes eux une gamme de lieurs encore plus large. La *restriction* qui déclare un *nom* locale ou privé à l'intérieur d' un terme et l'*imput* du π-calcul en sont des exemples. L' interaction entre ces lieurs, notamment dans le cas de calculs ou les transitions sont étiquetées (π-calcul), rend leurs traitement plus délicat que dans le cas du λ-calcul.

Des solution existantes ...

Différentes méthodes existent pour résoudre le problème des lieurs lors de leur spécification formelles. Aucune traitée jusqu' ici ne donne entièrement satisfaction. La méthode qui consiste a utiliser des indice de *de Bruijn* est bien maîtriser. Elle consiste à repérer les variables liées par des indices et non par des noms (ie le terme $\lambda x \lambda y\ xy$ est représenté par le terme *de Bruijn* $\lambda\lambda\ 10$). Elle possède le désavantage de générer des spécifications formelles très éloignée de leurs homologues informels. De plus, les développement réalisés avec une méthode de *de Bruijn* se sont révélés très gourmand en termes de lemmes techniques intermédiaires, souvent pénibles à réaliser. Une autre méthode, proposée par R.Pollack et J.McKinna, introduit deux ensembles disjoint, pour chaque type de variables. Un ensemble serra utiliser pour les noms libres, l' autre pour les liés. Les spécifications formelles ainsi obtenues sont plus proches de leurs versions informelles. Cette partition en deux de chaque type de variable reste pourtant insatisfaisant pour deux raisons. Une telle formalisation reste guidée par les contraintes des outils et non par un véritable contenu sémantique; les preuves formelles réalisées restent assez éloignées de leurs versions informelles. Enfin, il existe des méthodes dites d' ordre supérieurs ou les lieurs des calculs que l' on cherche a spécifier formellement sont définis à l'aide de fonctions. Les méthodes d' ordre supérieures, étudiées entre autre par J.Despeyroux, ont l' avantage de résoudre tout les problèmes d' α-conversion. Pourtant, a ce jour aucun démonstrateur de théorèmes n' est basé sur la logique permettant pleinement leur mise en oeuvre. De plus, les expériences réaliser avec cette approche ont montrées que les preuves n' avaient pas toujours la même structures que leurs homologues informelles.

. . . à mes travaux de recherche.
La thèse que je suis en train de rédiger consister à étudier une méthode pour représenter les lieurs des calcules de façons a ce que les raisonnement formels soient le plus proches possibles de leur formes informelles. J' ai choisi d' approfondir une technique proposer en 1994 par R.Gordon pour la syntaxe d' un λ-calcul non typé. Cette méthode consiste à définir un ensemble de terme à l' aide d' un formalisme à la *de Bruijn* puis, d' identifier les termes du calcul que l' on cherche a formaliser avec un sous ensembles de nos termes sans indices de *de Bruijn* apparents.

J' ai choisi de formaliser le **concς-calcul** dans le système **Coq**. Ce calcul inventé par A.Gordon et P.hankin est un calcul concurrent avec certaines des primitives objets du ξ-calcul de A.Abbadi et L.Cardelli. Sa taille relativement important (11 constructeurs) et son bonne expressivité en faisait un candidat naturel pour évaluer le comportement sur des cas réels de cette technique de formalisation des lieurs. Au long de ma thèse j' ai montré que l' on pouvait renforcer les théorèmes d' inductions, de façons à internaliser le renomage des variables liées. Ainsi, avec cette technique et une fois une travail (non négligeable) au niveau des indices de *de Bruijn* effectué, les raisonnement fait sur machines suivent le même cheminement que leur correspondant informel. J' ai aussi montré que sous certaines conditions les théorèmes informels d' inversion restent applicablent pour les propriétés sur les termes qui formalisent le **concς-calcul**. On montres que ces conditions nécessaires pour dériver les théorèmes d' inversion, sont *naturelles* et qu' elle sont toujours assumées lors des raisonnement à α-conversion près lors des preuves informelles.

3 Publications et Présentations.

[1] J' ai présenter au workshop *BRA Types 99* à Lökeberg (Suède) une première formalisation dans le système **Coq** de la syntaxe du **concς-calcul** qui utilise cette technique pour représenter les lieurs. On peut trouver une présentation détailler de ce travail dans l'article «The concς-calculus syntax formalized up to α-conversion in COQ». Il est en cours de soumission pour les proceedings de *BRA Types 99* .

[2] Une étude complète de **concς-calcul** comprenant une formalisation de la syntaxe, de la sémantique et du typage ainsi qu' une preuve en **Coq** de la conservation du typage lors de l' évolution est faite dans «A formalization of a concurrent object calculus up to α-conversion» . Cette article est à paraître dans les actes de *"International Conference on Automated Deduction" (CADE-17).*

4 Enseignements en Informatique.

Vacations
J' ai été vacataire à l' Université de Nice Sophia Antipolis lors des années universitaire 1998-1999 et 1999-2000.

1998-99	TP	*Système d' exploitation* MsDos (Cours Gilles Trombetoni) IUT GTR, première année (36h)
1998-99	TD	*Objets Formels pour l'Informatique* (cours de J.M. Fedou) Deug MP2 (36h)
1998-99	TP-TD	*Tableur* avec *EXCEL* comme support (Cours Christine Garcia) IUT GEA, première année (36h)

1999-00	TP	*Initiation à l' algorithmique et à la programmation* avec C comme support (cours Philippe Collard) IUT GTR, première année (24h)
1999-00	TP	*Système d'exploitation* unix (Cours Gilles Trombetoni) IUT GTR, première année (42h)
1999-00	TP	*Algorithmique et Structures de Données* avec java comme support (cours de J.M. Fedou) Deug MASS2 (39h)
1999-00	TD	*Programmation Fonctionnelle* avec Scheme comme support (Cours de J.P Roy) en Deug MP1 (26h).

Descriptif des enseignements
Les TPs de *Systeme d' exploitation* avec MsDos et unix complétaient les cours d'introduction aux systèmes d' exploitation.
Les TDs-TPs de EXCEL apportaient aux étudiant une connaissance assez poussée (macros) du *tableur*.
Les TDs *Objets Formels pour l'Informatique* représentaient un survol rapide mais complet des bases de l'informatique théorique.
Les TPs d'*Algorithmique et Structures de Données* complétaient le cours dont les objectifs étaient de donner les notions de base de programmation impérative : structures de contrôles, appels à des procédures, récursivité. Les étudiants devaient connaître les bases de la programmation objet, les structures de données élémentaires, tableaux, listes chaînées, arbres ainsi que quelques algorithmes de base (tris, techniques "diviser pour régner", ...).
Les TPs de *programmation fonctionnelle* sont une solide introduction à la programmation fonctionnel avec le langage Scheme comme support.

5 Enseignements en Mathématiques.

Coopération
J' ai été coopérant du service national durant les années universitaires 1994-1995 et 1995-1996 à l' université de Bangui (RCA). Durant ces deux année, je fus rattaché au laboratoire de mathématiques de l' université des sciences. Ma tache principale fut d' assurer les enseignement de mathématiques de première années des DEUGs scientifiques.

1994-96	cours et TD	*Algèbre*	MP1
1994-96	cours	*Analyse*	MP1
1994-96	cours	*Mathématiques*	PC1
1994-96	TD	*Analyse*	MP2
1995-96	cours et TD	*Géométrie*	IUT, première année

Descriptif des enseignements
Le contenu des cours de mathématiques est sensiblement le même que ceux enseigner dans les université françaises. J' ai utilisé mes notes de cours de classe préparatoire aux grandes écoles comme support de cours ainsi que les ouvrages classique de référence (précis ...). En section *physique-chimie* (PC) le contenu du cours étant moins danse que celui de *maths-physique* (MP) tout l' enseignement en mathématique était regroupé sur un seul module. Le cours de *Géométrie* avait pour objectif d' apporter aux étudiants les bases de la géométrie plane.