Rapport INRIA 1994 — Programme 2

Spécifications et preuves de programmes

PROJET COQ

3 mai 1995

PROJET COQ

Spécifications et preuves de programmes

Localisation : *Rocquencourt*

Mots-clés : calcul formel (1), déduction automatique (1), lambda-calcul (1), langage fonctionnel (1), logique (1), preuve de programme (1), spécification formelle (1), synthèse de programme (1), typage (1).

1 Composition de l'équipe

Responsables Scientifiques

Gérard Huet, DR INRIA
Christine Paulin, CR CNRS, ENS Lyon

Chercheurs confirmés

Philippe Audebaud, ENS Lyon
Gilles Dowek, CR INRIA
Pascal Manoury, Post Doc INRIA jusqu'au 30-9
Claude Marché, Post Doc INRIA jusqu'au 30-9
Chet Murthy, Chateaubriand+INRIA jusqu'au 30-7
Erik Poll, Post Doc HCM (ENS Paris) à partir du 15-11
Benjamin Werner, CR INRIA

Chercheurs de 3ème cycle

Samuel Boutin, Ater CNAM
Cristina Cornes, Bourse INRIA
Judicaël Courant, ENS Paris
Jean-Christophe Filliâtre, ENS Paris

Eduardo Gimenez, Bourse INRIA
Hugo Herbelin, Paris 7
Valérie Ménissier-Morain, Ater Évry
César Muñoz, Bourse du gvt Colombien + INRIA
Catherine Parent, Dassault, ENS Lyon
Amokrane Saïbi, Bourse du gvt Algérien + INRIA

Stagiaires

Chrystel Barraband, Paris 7
Hassen Saïdi, Paris 7
Daniel Hirschkoff, Paris 6/École des Ponts

Secrétariat (commun avec les projets CRISTAL et PARA)

Ghislaine Le Corre, INRIA
Sylvie Loubressac, INRIA

2 Présentation du projet

Le projet Coq s'intéresse à la conception d'environnements de manipulation de preuves formelles adaptés au développement de programmes certifiés, et plus généralement à la problématique de recherche en théorie des types.

La théorie des types, qui remonte aux travaux de Russell sur Principia Mathematica, puis de Church sur la logique d'ordre supérieur, a été profondément renouvellée par l'approche constructive des mathématiques, et en particulier les travaux du logicien suédois Per Martin Löf. Le λ-calcul, fondement des langages de programmation fonctionnels, est également la structure combinatoire de base de la théorie de la démonstration de la logique intuitionniste présentée par la déduction naturelle. Le principe de correspondance entre programmes et preuves constructives, ou isomorphisme de Curry-Howard, raffiné par les diverses interprétations de réalisabilité, dresse un cadre général à un programme de développement de logiciels certifiés à partir d'une analyse logique de leurs spécifications.

Dans ce cadre général, les projets Formel puis Coq ont poursuivi des recherches pendant les 10 dernières années, qui ont abouti aux résultats suivants :

- Une infrastructure logique a été définie, autour du formalisme dit "Calcul des Constructions Inductives", dont les propriétés méta-théoriques essentielles ont été étudiées.
- Un langage de spécification de très haut niveau, Gallina, permet d'exprimer des axiomatisations logiques et des spécifications algorithmiques, mélangeant un calcul des prédicats muni de types inductifs avec un mécanisme de définitions autorisant la récursion et la définition de familles inductives.
- Un mécanisme de compilation d'axiomes de récurrence adaptés au raisonnement sur les structures inductivement définies autorise une grande souplesse dans les principes de preuve.
- Un système d'aide aux démonstrations, programmable par tactiques, permet de donner une assistance à la recherche de preuves.
- Une tactique spécialisée permet de prouver la correction d'un programme fonctionnel vis-à-vis de ses spécifications fournies comme annotations.

Ces travaux de recherche ont été concrétisés dans un logiciel Coq suffisamment robuste pour le développement de preuves conséquentes par une communauté d'utilisateurs dans un large spectre d'applications, sur une quinzaine de sites.

Le projet Coq regroupe deux équipes géographiques travaillant en étroite concertation : à l'INRIA-Rocquencourt, et au laboratoire LIP de l'École Normale Supérieure de Lyon. Le projet Coq a été reconnu comme projet commun CNRS-INRIA en Octobre 1993.

Les points marquants de l'année 1994 sont les suivants:

- Une nouvelle version de distribution de Coq, autorisant la compilation de théories, l'écriture de tactiques par les utilisateurs, et l'extension de la syntaxe par des notations spécifiques a été mise au point; il est prévu de la distribuer par ftp en β-test vers la fin de l'année.
- Un algorithme d'unification d'ordre supérieur utilisant les substitutions explicites a été mis au point.
- Un certain nombre de procédures de décision ont été implémentées, augmentant significativement la puissance d'automatisation des raisonnements.

- Un programme de formalisation constructive de la théorie des catégories a été démarré.

3 Actions de recherche

3.1 Le Calcul des Constructions Inductives

Le Calcul des Constructions Inductives est un langage de spécification très général, qui étend le Calcul des Constructions, défini par Thierry Coquand en 1985, et étudié dans le projet depuis comme formalisme de base pour la représentation formelle de notions mathématiques et informatiques. Il comprend un fragment logique, définissant un langage de propositions, étendant la théorie des types de Church, mais aussi un langage de preuves formelles, sorte de λ-calcul typé généralisant les structures Automath. Il comprend également un fragment calculatoire, comprenant le λ-calcul polymorphe $F\omega$ de Girard, avec des types dépendants, et la possibilité de déclarer des notions inductives, généralisant la théorie constructive des types de Martin-Löf.

Ce langage de spécifications peut être vu de manière moins ésotérique comme une combinaison harmonieuse du calcul des prédicats d'ordre supérieur, de spécifications relationnelles à la PROLOG, et de définitions récursives à la ML. C'est donc un système de typage donnant un fondement logique bien compris au mélange de programmation logique et de programmation fonctionnelle.

Nous développons un environnement interactif de manipulation de preuves appelé Coq, qui permet à l'utilisateur d'écrire des spécifications, de les prouver correctes, et d'extraire de façon mécanique des programmes les réalisant. Ce système est écrit en Caml-Light, et les programmes extraits sont exécutables en Caml-Light. Il est donc clair qu'il y a de nombreuses interactions avec le projet Cristal. Le but à long terme est bien sûr de développer un environnement de construction de programmes certifiés, combinant les efforts Coq et Cristal.

3.1.1 Méta-théorie

Participant : Benjamin Werner

Benjamin Werner a poursuivi son travail concernant l'étude méta-théorique du Calcul des Constructions Inductives (CCI). Dans sa thèse

[Wer94], soutenue en Mai, il a prouvé la normalisation forte et la confluence de CCI, qui est le cœur du système Coq. Ces résultats sont essentiels puisqu'ils garantissent la cohérence logique du système et la décidabilité de la relation de typage. Le sytème étudié inclue la η-conversion, ce qui rend la propriété de Church-Rosser non triviale.

B. Werner a approfondi ce dernier point en étudiant plus en détail les relations entre Church-Rosser et d'autres propriétés de λ-calculs typés, et en particulier la normalisation. Ces travaux sont exposés dans un article écrit en commun avec Herman Geuvers de l'université d'Eindhoven et présenté au neuvième symposium LICS en Juillet à Paris[21].

3.1.2 Définitions récursives

Participants : Pascal Manoury, Chrystel Barraband

Le travail de Pascal Manoury est centré autour de l'extraction de programmes. Le but de sa venue, en tant que boursier post-doc au sein du projet Coq, a été de porter dans le système Coq l'algorithme d'extraction de programmes qu'il avait développé conjointement avec Marianne Simonot au cours de leurs études doctorales.

Cet algorithme permet d'extraire un programme (λ-terme) en prouvant qu'une fonction définie par un système d'équations est totale. L'intégration de cet algorithme dans le système Coq fournit à l'utilisateur un moyen simple et naturel de définir des fonctions. Concrètement, au lieu d'écrire directement un λ-terme, l'utilisateur a désormais la possibilité de définir ses fonctions par filtrage (*pattern matching*) à la manière dont cela est fait dans les langages ML.

Ce travail a nécessité que Pascal Manoury se familiarise d'une part avec le Calcul des Constructions, afin de "porter", au niveau théorique, des résultats établis pour un autre système de λ-calcul typé (la Théorie des Types Récursifs de M. Parigot); d'autre part, Pascal Manoury a du se familiariser avec le système Coq afin d'y implémenter à son tour son algorithme.

Cette étude a abouti à l'adjonction au système Coq d'une nouvelle clause de définition : les `Recursive Definition`. Cette nouvelle clause fait partie de la nouvelle version de Coq.

Du point de vue de l'utilisateur, les choses se présentent de la manière suivante. Actuellement, l'utilisateur peut introduire des types de données inductifs, tels que les entiers unaires qu'il déclare par :

```
Inductive Set nat = O : nat | S : nat->nat.
```

Le système Coq donne alors la possibilité de manipuler des fonctions définies suivant un schéma de définition primitif récursif, en fournissant un opérateur adéquat. Dans le cas des entiers, on dispose d'un terme `nat_rec` tel que :

$$(\texttt{nat\_rec}\ f\ g\ O) = f \quad \text{et} \quad (\texttt{nat\_rec}\ f\ g\ (S\ n)) = (g\ n\ (\texttt{nat\_rec}\ f\ g\ \ n))$$

même si ce schéma est suffisant d'un point de vue théorique, il est très lourd et peu naturel à utiliser.

Les travaux de Pascal Manoury permettent d'engendrer automatiquement un programme à partir d'une déclaration équationnelle de la fonction à définir. Par exemple, une fonction testant pour deux entiers n et m si n est plus petit que m peut s'écrire dans son formalisme:

```
Function Definition inf : nat->nat->bool =
     O     m      => true
   (S n)   O      => false
   (S n) (S m)    => (inf n m)
```

Cette définition comprend trois éléments.

a) La déclaration d'une nouvelle variable; ici `inf`.

b) La déclaration du type de cette variable :
`nat->nat->bool`.

c) Les équations que doivent satisfaire la fonction visée ;
`inf(O,n) = true`, `inf((S n),O) = false`
et `inf((S n),(S m)) = inf(n,m)`.

Cette déclaration concise peut être comparée au terme de Coq engendré par l'algorithme et qui dans la version V5.8 doit être explicitement écrit par l'utilisateur :

```
[x1:nat](<nat->bool>Match_rec x1 with
              [x2:nat]true
              [x1_1:nat][H1_1:nat->bool][x2:nat]
                (<bool>Match_rec x2 with
                       false
                       [x2_1:nat][H2_1:bool](H1_1 x2_1))).
```

Le terme construit est bien du type attendu et, de plus, satisfait les équations. C'est donc un *programme* qui calcule la fonction.
Pascal Manoury a également adapté son algorithme pour pouvoir l'appliquer à des types de données polymorphes tels que les listes. L'utilisateur peut ainsi, de manière analogue, définir la fonction de concaténation de deux listes d'objets d'un type X arbitraire :

```
Function Definition append
      : (X:Set)(list X)->(list X)->(list X) =
 (Nil X)       ys => ys
 (Cons X x xs) ys => (Cons X x (append xs ys)).
```

Cette nouvelle possibilité enrichit la *chaîne de certification de programmes* de Coq à deux points de vue. D'une part, elle permet de définir de façon naturelle des programmes. D'autre part, couplée avec la tactique `Program` développée à Lyon, elle augure d'une large automatisation de la preuve de spécification.

La généralisation de la compilation ProPre à des récursions générales a été étudiée par Chrystel Barraband dans son stage de DEA[28].

3.1.3 Inversion des définitions inductives

Participants : Cristina Cornes, Chet Murthy

Cristina Cornes est en thèse depuis octobre 1993. Cette année, elle a implanté dans Coq des tactiques pour démontrer automatiquement les lemmes de la forme `~O=(S n)` (non-confusion des constructeurs) et `(S n)=(S m)->n=m` (lemmes d'injection des constructeurs d'un type inductif).

Elle a aussi implanté l'algorithme proposé pendant son stage de D.E.A. pour fabriquer, à partir de la définition d'un prédicat inductif, plusieurs énoncés d'inversion donnant à l'utilisateur la possibilité d'indiquer où il veut faire le filtrage.

Son travail s'oriente maintenant vers l'étude des relations entre ProPre et l'inversion pour faire l'analyse de définitions par motifs avec types dépendants.

En parallèle, Chet Murthy a implémenté dans la version 5.10 des tactiques d'inversion donnant satisfaction dans un grand nombre de situations.

3.1.4 Egalité

Participant : Claude Marché

Claude Marché est boursier post-doc au sein du projet Coq depuis le 1er octobre 1993. Il a apporté l'expérience de ses travaux de thèse dans le domaine du traitement de l'égalité par les techniques de réécriture pour comprendre et améliorer le traitement de l'égalité dans le système Coq. Il a d'autre part continué sa recherche théorique sur les techniques de réécriture [25, 24]

Les techniques de réécriture doivent permettre d'automatiser une partie des preuves de manière à, d'une part accélérer le travail de développement de l'utilisateur, et d'autre part permettre des développements de plus en plus évolués. En effet, Coq reste actuellement un système très interactif, et il s'agit d'automatiser une partie du travail effectué actuellement par l'utilisateur, en particulier au niveau des preuves mettant en jeu le prédicat d'égalité. L'enjeu est important car si l'on veut que le système Coq puisse devenir un outil de développement réellement utile en pratique, il est indispensable de rendre automatiques toutes les tâches qui peuvent l'être.

Cadre du problème et résultats déjà obtenus Dans le domaine de la spécification et la preuve de programmes, on utilise de manière privilégiée des spécifications *algébriques*, où l'on décrit des types de données par des constructeurs, des fonctions et des prédicats manipulant ces données, et l'on spécifie par des équations éventuellement conditionnelles les relations entre les constructeurs et les axiomes vérifiés par les fonctions et prédicats. Ensuite, les preuves que l'on établit sur de telles spécifications mettent en jeu de manière privilégiée les techniques de réécriture.

Le problème qui se pose alors pour Coq est de traduire un tel formalisme dans le calcul des constructions, et ceci étant fait d'interpréter les preuves par réécriture en des séquences de tactiques de preuves pour la machine constructive.

Claude Marché a commencé ce travail de traduction pendant cette année de bourse. Nous savons traduire automatiquement une spécification algébrique avec éventuellement relations entre constructeurs, la cible étant ce qu'on appelle un *sétoïde*, c'est-à-dire un type muni d'un prédicat d'égalité satisfaisant les propriétés de congruence vis-à-vis des constructeurs.

Pour la définition de fonctions et de prédicats par équations sur l'algèbre ainsi définie, la cible choisie est une définition d'un objet du type approprié et d'axiomes correspondants aux équations. La partie maintenant plus difficile à partir de là est de réellement construire une *application* entre sétoïdes, vérifiant en particulier les propriétés de compatibilité par rapport aux constructeurs. Nous ne savons pas faire cette opération entièrement mécaniquement et vu la difficulté que cela représente (nécessité de preuve par récurrence) nous envisageons seulement une technique semi-automatique.

Pour l'instant est implémentée dans une version expérimentale de Coq une tactique de normalisation par rapport à un système de réécriture du premier ordre, ceci pour un prédicat d'égalité arbitraire sous la condition d'existence de l'axiome d'associativité correspondant ainsi que les axiomes de congruence pour chacun des opérateurs de premier ordre apparaissant dans le système de réécriture. Cette implantation a révélé un sérieux problème pratique à savoir la lourdeur de la traduction de la preuve de réduction en un λ-terme.

3.1.5 Substitutions explicites et unification

Participant : Gilles Dowek

En collaboration avec Th. Hardin de l'Université de Paris 6 et C. Kirchner de l'INRIA-Lorraine, G. Dowek a étudié un nouvel algorithme d'unification à l'ordre supérieur utilisant le formalisme des substitutions explicites. L'utilisation de ce formalisme a permis de simplifier la gestion des variables liées qui était le point délicat des algorithmes classiques. Reformulé dans le calcul des substitutions explicites, le problème de l'unification à l'ordre supérieur apparrait comme un problème d'unification équationelle du premier ordre. Ce travail est presenté dans [18]

3.1.6 Liens entre théorie des types et théorie des ensembles

Participant : Gilles Dowek

Un problème central pour les systèmes de traitement de preuves mathématiques est celui du langage dans lequel ces mathématiques s'expriment. Depuis le début de ce siècle deux traditions s'oppose. La *théorie des types* issue des travaux de Whitehead, Russell, Church, Martin-Löf,

etc. et la *théorie des ensembles* issue des travaux de Zermelo, Frænkel, Von Neumann, Bernays, *etc.* Il est frappant de constater que si la théorie des ensembles domine largement la théorie des types sur le terrain des mathématiques informelles (au point que la théorie des types était considérée comme quasiment morte à la fin des années soixante) la plupart des systèmes de traitement de preuves sont quant à eux basés sur la théorie des types. Il semble de ce fait important de comprendre les différences profondes entre ces deux théories, ainsi que les raisons qui expliquent le relatif échec de la théorie des ensembles pour les applications informatiques (démonstrations automatique, mathématiques constructives, utilisation comme langage de spécification, etc.).

Une des premières différences entre les deux théories est que la théorie des types donne en général un langage explicite pour exprimer les objets mathématiques alors que la théorie des ensembles ne donne que des axiomes d'existence pour ces objets. Gilles Dowek a généralisé un résultat de L. Henkin montrant l'équivalence de deux formulations de la théorie des types: l'une avec un langage d'expression des objets mathématique (λ-calcul) et l'autre avec un schéma d'axiome exprimant l'existence de ces objets. L'aspect frappant de cette démonstration est que, quand on applique la méthode habituelle pour rendre explicites les notations implicitement contenues dans des axiomes d'existence (skolemization), on obtient non le λ-calcul, mais un langage voisin basé sur la notion de combinateur. La preuve d'équivalence de ces deux langages montre qu'à la différence des formulations de la théorie des types basées sur le λ-calcul, celles basées sur un schéma d'axiome d'existence ou un langage de combinateurs ne peut faire l'économie de la notion de fonction de plusieurs variables. Une formulation de la théorie des ensembles avec notation explicite pour les objets, a été obtenue par la même méthode, cette formulation évoque le langage des combinateurs et l'équivalence de ce langage avec un autre plus proche du λ-calcul est laissée ouverte. Ces résultats sont exposés dans [20].

La deuxième étape de ce travail a été une formulation de la théorie des types dans laquelle les types ne sont que des ensembles particuliers. Cette formulation semble combiner certains avantages de la théorie des types (notation explicite pour les objets, fonctions primitives, propositions comme objets) et de la théorie des ensembles (formulation au premier ordre, notion unique de collection). Elle a permis de simplifier les démonstrations de résultats classiques sur la théorie des types (théo-

rème de Henkin, théorème de Miller). Néanmoins, elle souffre encore de deux faiblesses: la possibilité de former des propositions non significatives et la nécessité d'une vérification dynamique des types lors de la réduction. Ce travail est présenté dans [31].

3.1.7 Alternative au Calcul des Constructions Inductives

Participant : Philippe Audebaud

Le Calcul des Constructions Inductives, dans l'approche actuelle, utilise plus ou moins explicitement un système de sortes conjuguant une hiérarchie d'univers prédicatifs et un marquage du contenu informatif des termes (*Prop*, *Set*). Par ailleurs, il donne lieu à un mécanisme complexe de schémas d'élimination. On propose un système multisorté généralisant le système *ECC* de Luo permettant de tirer le meilleur parti du mécanisme d'inclusion d'univers, à la fois dans la justification théorique des schémas d'élimination valides, et dans leur mise en œuvre pratique. Ce travail est développé en trois étapes: (1) extension du système de sortes avec cumulativité, (2) extension en un système avec points fixes, et (3) extension prenant en compte les types inductifs. Le point (1) est terminé : le système a toutes les bonnes propriétés méta-théoriques souhaitées. Les points (2) et (3) sont décrits en tant que systèmes formels. Leurs propriétés méta sont en cours de développement. Cette étude a pu être enrichie récemment de la suggestion de C. Paulin de prendre en compte l'inclusion de *Prop* dans *Set*, mécanisme important non capturé par la cumulativité d'univers.

3.1.8 Types mutuellement inductifs

Participant : Christine Paulin

Christine Paulin a travaillé cette année à la définition et l'implantation d'une nouvelle structure de type inductifs dans Coq. Cette nouvelle structure autorise les définitions mutuellement récursives ce qui constitue une amélioration essentielle par rapport à la version précédente. Pour assurer la cohérence du système, les définitions inductives ne peuvent être acceptées que si elles satisfont une condition de monotonicité assurée par une restriction syntaxique de positivité. Cette condition a été étendue pour autoriser des occurrences emboitées dans des définitions

inductives. Par exemple, l'occurrence de X dans $A \wedge X$ ou ($\texttt{list}X$) est positive.

Une autre différence majeure est la séparation du schéma d'élimination des définitions inductives, qui permet d'effectuer une récurrence structurelle ou d'exprimer un principe de minimalité en deux schémas "plus simples". L'un est une analyse par cas sur les différents constructeurs de la définition inductive. L'autre est une définition par point fixe "gardé". C'est-à-dire que certaines restrictions sont imposées sur les occurrences des appels récursifs afin de maintenir la terminaison des programmes et la cohérence du système.

Cette séparation en deux schémas a l'avantage de bénéficier de règles de typage et de réduction simples (seule la vérification de la condition de garde reste compliquée). De plus, elle correspond naturellement à la manière dont les fonctions sur les types de données sont définies dans les langages fonctionnels. D'autre part, cette décomposition permet d'accepter simplement des types inductifs avec une notion plus large de positivité. Un tel système avait été proposé par Th. Coquand dans le cadre du système ALF, les conditions ont été adaptées au cas du Calcul des Constructions et implémentées dans Coq. Ce travail est décrit dans [38].

3.1.9 Types coinductifs

Participant : Eduardo Giménez

Eduardo Giménez travaille à l'incorporation dans Coq des types contenant des objects potentiellement infinis, dits types co-inductifs. L'exemple typique est le type des suites infinies (streams) des langages fonctionnels avec évaluation paresseuse. Ces types peuvent être utilisés pour modéliser la notion d'entrée/sortie dans les langages fonctionnels ainsi que des systèmes de calcul parallèle comme des réseaux de G. Kahn et D. Mac Queen.

Une représentation possible des streams dans Coq avait été étudiée antérieurement par F. Leclerc et C. Paulin. Cette représentation etait basée sur la codification de plus grands points fixes (utilisant le polymorphisme et l'imprédicativité) et l'utilisation des opérateurs co-récursifs.

Durant l'année dernière, E. Giménez a étudié une forme plus simple et plus directe de représentation des types co-inductifs, dans l'esprit des travaux de C. Paulin sur les types mutuellement inductifs. Il a formalisé

une extension du Calcul des Constructions avec des types co-inductifs primitifs. Dans ce calcul, la définition des objects infinis est basée sur une condition de "garde" proposée par T. Coquand, similaire à celle utilisée actuellement pour la récursion sur les types inductifs. Il a montré que le principe proposé par T. Coquand est insuffisant pour garantir la normalisation du système dans le cas de types co-inductifs utilisant une quantification de deuxième ordre, et il a proposé une modification de la condition pour résoudre ce problème. Une méthode pour interpréter des définitions gardées utilisant des opérateurs co-récursifs a été developpée afin de justifier cette nouvelle condition et d'assurer la cohérence logique du système. Un article décrivant cette méthode a été presenté à la conférence annuelle du projet "Types".

E. Giménez travaille actuellement à l'implémentation de cette représentation des types co-inductifs.

3.1.10 Types quotients

Participant : Samuel Boutin

Samuel Boutin a travaillé, dans le cadre de sa thèse, à la formalisation de "types quotients" permettant d'augmenter l'expressivité du Calcul des Constructions avec types Inductifs.

Ses travaux sont inspirés d'un article de Backhouse et. al. intitulé "Do-it-yourself type theory". Dans cet article, les auteurs décrivent informellement une manière de definir des structures quotients dans la théorie des types de Martin-Löf. Samuel Boutin a formalisé les types quotients dans le Calcul des Constructions Inductives. Il a implémenté les structures quotients dans une version non distribuée du système Coq.

Illustrons ses travaux par l'exemple des "bags". Un "bag" est un ensemble avec répétition possible des éléments. On peut définir les "bags" sur des objets d'un type arbitraire, dans le système Coq, à l'aide d'une structure quotient: Etant donné une égalité `Eq_Prop`,

```
Quotient Bag[A:Prop] : Prop :=
  empty : (Bag A)
| cons : A -> (Bag A) -> (Bag A)
Equality Eq_Prop
Equations
  (a:A)(b:A)(bag:(Bag A))
         (cons a (cons b bag))=(cons b (cons a bag)).
```

Informellement, on définit donc les "bags" polymorphes comme des listes polymorphes (avec comme uniques constructeurs `empty` et `cons`) en rajoutant une condition (une équation du premier ordre) qui nous assure que l'on peut intervertir les deux premiers éléments de "la liste". Cette condition permet en fait d'intervertir deux eléments quelconques.

Un type quotient est donc fabriqué sur le même modèle qu'un type inductif: à l'aide de constructeurs. Il en diffère par une liste d'équations entre ces constructeurs. En effet, on formalise maintenant des algèbres initiales.

Comme pour les types inductifs, le système engendre automatiquement un schéma d'élimination inspiré de celui des types inductifs. Voici le schéma d'élimination non dépendant des bags:

```
Constant Bag_ind:
(A,P:Prop) P -> (rec1:A->(Bag A)->P->P)
    (Rec_bag:P)(a,b:A)(bag:Bag A)
    (Eq_Prop P (rec1 a (cons A b bag)(rec1 b bag Rec_bag))
               (rec1 b (cons A a bag)(rec1 a bag Rec_bag)))
                  ->(Bag A)->P.
```

A l'aide de ce schéma il est possible de fabriquer des fonctions sur les bags. L'implémentation de Samuel Boutin ne permet pour l'instant que la génération automatique de schémas d'élimination non dépendants. La définition des schémas d'élimination dépendants permettant de définir en particulier des prédicats sur les types quotients est le prolongement naturel de ce travail.

3.1.11 Logique Classique

Participant : Philippe Audebaud

Le traitement des exceptions, des échappements, etc est récemment apparu comme relevant du cadre de la logique classique. Du point de vue mathématique, le mu-calcul proposé par M. Parigot est un formalisme solidement justifié théoriquement, et particulièrement bien adapté aux systèmes en déduction naturelle. Philippe Audebaud a proposé son adaptation au Calcul des Constructions (originel). Cette extension s'est avérée aisée, cependant, son incidence au niveau des structures de données requiert davantage de travail.

3.1.12 Fondement des λ-calculs typés et de la prouvabilité

Participant : Hugo Herbelin

Hugo Herbelin a mis en évidence que les λ-calculs typés ne sont pas moins liés aux calculs de séquents du type des calculs LJ ou LK de Gentzen qu'ils ne le sont aux systèmes de déduction naturelle. De fait, il a montré qu'on obtient un isomorphisme entre ce genre de calcul et le λ-calcul pour peu qu'on altère légèrement la syntaxe habituelle des λ-termes applicatifs, obtenant alors une structure de λ-calcul comparable à la structure des machines à environnements.

Cet isomorphisme avec le λ-calcul, valable à la fois dans un cadre intuitionniste et dans un cadre classique, est un atout pour les calculs du type de LJ ou LK qui bénéficiaient déjà de bonnes propriétés concernant la démonstration automatique. La question se pose de savoir s'il n'est pas plus justifié de baser l'implémentation d'une théorie des types sur ce genre de calcul plutôt que sur la déduction naturelle (bien que pour une description intelligible de la théorie des types, la déduction naturelle reste sans doute préférable, justement pour son côté "naturel").

La preuve d'une formule dans un certain système logique peut être interprétée comme une stratégie gagnante pour un certain jeu à deux joueurs se disputant la validité de la formule. L'analogie est spécialement mise en valeur lorsqu'on considère des calculs de séquents avec coupures du type des calculs LJ ou LK de Gentzen puisque ces calculs induisent directement des jeux pour lesquels les stratégies gagnantes sont isomorphes aux preuves. Exploitant cet isomorphisme, il est possible de définir une alternative à la normalisation des preuves (= élimination des coupures) en faisant jouer les stratégies gagnantes correspondantes l'une contre l'autre. Hugo Herbelin a montré que cette interaction entre stratégies gagnantes revenait à éliminer les coupures selon une stratégie de réduction faible de tête.

Mélangeant ces deux résultats, il est possible d'obtenir un modèle syntactique du λ-calcul typé, recourant à des notions de jeu et de stratégie gagnante similaires à celles décrites récemment par Abramsky *et al.*

3.1.13 Unification arithmétique

Participant : Hassen Saïdi

Au cours de son stage de DEA, Hassen Saïdi a étendu l'algorithme d'unification à l'ordre supérieur à un λ-calcul avec un opérateur de récurrence (le système T de Gödel). Cette approche est une alternative à l'utilisation de l'algorithme ProPre de P. Manoury et M. Simonot pour définir des fonctions "par filtrage". Les rapports entre ces deux approches restent à être étudiés.

3.2 Le système Coq

3.2.1 Une nouvelle architecture: Coq V5.10

Participant : Chet Murthy

Durant les 6 premiers mois de l'année, Chet Murthy a développé une nouvelle architecture pour Coq, dont les traits saillants sont les suivants:

- modules vérifiables séparément;
- syntaxe extensible;
- tactiques utilisateur;
- synthèse de types;
- arbres de preuves.

Une description plus détaillée de certains points techniques est développée ci-dessous

- Le système entier a été rendu davantage fonctionnel, ce qui permet un contrôle plus fin de l'impact de la modification d'un module.
- La structure des termes dans la version 5.8 consistait en trois différents types de données gérés par un schéma de liaison à la "de Bruijn". Chet Murthy les a unifiés en un seul type générique de terme, pour lequel les fonctions les plus courantes (e.g. substitution, parmi d'autres) ne sont écrites qu'une fois. Deux autres types de données, qui étaient gérés avec un schéma de nommage explicite, sont actuellement unifiés en un deuxième type de donnée. Ces deux types, l'un étant lié à la de Bruijn, l'autre, avec des noms explicites, sont des jumeaux, et, de la même façon que toutes les opérations internes comme la substitution ne sont faites qu'une

fois, toutes les opérations externes, comme l'écriture et la lecture, sont faites avec ce deuxième type de donnée.

- Chet Murthy a remplacé les divers moyens de lecture par un lecteur interprétatif, qui fonctionne sur une grammaire LL(1), et a également remplacé les diverses fonctions d'écriture par un paragrapheur paramétré. Les données de ces deux modules sont chargées au moment de l'exécution.

- Le noyau du système, qui (1) gère les arbres de preuves partiels et complets, qui (2) fait l'inférence et la vérification des types, et qui (3) gère la portée, les types, et les valeurs des identificateurs libres, a été coupé selon ces trois parties, de façon à ce qu'on puisse (avec un degré de sécurité raisonnable) en réécrire une partie, indépendamment des autres.

- La partie du système qui gère les arbres de preuves a été réécrite pour permettre une granularité dans la structure des preuves. Cette granularité est utilisée pour stocker la liste des commandes de l'utilisateur qui a construit cette preuve, et peut être également utilisée pour re-exécuter la preuve, avec une détection et localisation automatique des erreurs (suite, par exemple, à une modification incorrecte d'un processus de démonstration heuristique).

- Ce sous-système de gestion des arbres de preuves permet de faire des développements multiples indépendants en parallèle. Il permet également d'attaquer séparément des sous-obligations différentes d'un développement. L'arbre de preuve est vu comme un document en cours d'écriture, et l'environnement garantit la maintenance de cohérence de l'arbre.

- Des variables existentielles ont aussi été integrées dans ce système, permettant à l'utilisateur de construire une preuve en gardant indéfinies certaines parties de la preuve, qui seront specifiées plus tard.

- Le sous-système d'inférence et de vérification de type est actuellement presque indépendant de tout le reste du système ; les seules données dont il a besoin sont les valeurs et les types des identificateurs connus globalement. Ceci permet des modifications et le prototypage d'extensions au système de types.

- Dernièrement, un système de modules permettant la vérification séparée de développements a été réalisé. Son utilisation raccourcit considérablement le chargement des bibliothèques nécessaires à un développement, ces bibliothèques étant maintenant chargées comme des objets compilés.

3.2.2 Développement d'une version de distribution

Participants : Gérard Huet, Christine Paulin, Pascal Manoury, Benjamin Werner, Cristina Cornes, Judicaël Courant, Jean-Christophe Filliâtre, César Muñoz, Catherine Parent, Amokrane Saïbi

Une grande partie de l'été et de l'automne ont été consacrés à la mise au point d'une nouvelle version de distribution du système, à partir de l'architecture de la V5.10. La modification la plus conséquente a été le remplacement des structures de représentation des types inductifs par une nouvelle structure autorisant les récursions croisées, comme documenté plus haut. Cette modification a entraîné à son tour des modifications majeures de la tactique Program par exemple. Judicaël Courant et Jean-Christophe Filliâtre ont systématisé le rattrapage d'erreur et ont modifié le compilateur de modules pour qu'il accepte à la volée les commandes de notations. César Muñoz a commencé a réorganiser les bibliothèques de tactiques. Cristina Cornes a pris en charge les règles d'impression du système de base et de ses bibliothèques principales, tandis qu'Amokrane Saïbi écrivait les règles d'analyse syntaxique correspondantes.

Un effort de documentation du nouveau système a été entrepris. G. Huet a commencé l'écriture d'un manuel introductif, P. Manoury a mis en chantier un nouveau manuel de référence, et de nombreuses documentations spécialisées ont été écrites, telles qu'un manuel sur l'écriture de tactiques par l'utilisateur, écrit par J.C. Filliâtre.

Depuis son retour à Rocquencourt, B. Werner a participé à l'effort de mise au point de la version 5.10 de Coq. Il est le coordinateur des développements autour de cette version. Il a en particulier écrit la procédure d'installation du système en vue de sa diffusion.

Une distribution d'une version beta-test du nouveau système est prévue pour la fin de l'année.

Une communauté d'utilisateurs de Coq s'est maintenant constituée sur une quinzaine de sites : Rocquencourt, Lyon, Sophia, CNAM, Nancy,

Dassault-Suresnes, Bordeaux, Montréal, CNET-Lannion, CWI, Utrecht, Eindhoven, Bell Labs, Copenhague, Udine. Une "hotline" électronique est gérée par le projet, pour répondre rapidement aux difficultés des utilisateurs. Nous avons créé un Club Coq électronique, permettant aux utilisateurs de partager leurs remarques, difficultés, et autres informations. Le modérateur en est B. Werner. Une zone CONTRIB de la distribution permet de partager les développements effectués par les utilisateurs. G. Huet en est le coordinateur.

3.2.3 Une procédure de décision pour la logique propositionnelle intuitionniste

Participant : César Muñoz

Il est bien connu que la logique propositionnelle, classique et intuitionniste, est décidable. Pour le cas classique, un algorithme de décision simple et facile à implanter consiste à vérifier les valeurs de vérité d'une formule, en utilisant les tables de vérités. Néanmoins, les tables de vérités ne sont pas un modèle pour la logique intuitionniste, donc cette méthode n'a pas de sens dans ce cadre.

Le travail de César Muñoz, pendant son stage de DEA, a porté sur l'étude des différents systèmes de preuve et des méthodes de décision pour la logique propositionnelle intuitionniste. Il a proposé une interprétation de l'isomorphisme de Curry-Howard pour le calcul des séquents avec plusieurs formules à droite LJT* (présenté par Dyckhoff). Cette interprétation est la base d'une procédure de décision et d'une procédure de simplification de séquents, implantées comme des tactiques (`Tauto` et `Intuition`) dans le système Coq.

3.2.4 Une procédure de décision pour le calcul des prédicats direct

Participant : Jean-Christophe Filliâtre

Le Calcul des Prédicats Direct est le fragment du Calcul des Prédicats du premier ordre prouvable dans le Calcul des Séquents Classique de Gentzen sans utiliser la règle de contraction. Il s'avère que ce calcul est décidable et J. Ketonen and R. Weyrauch en ont donné une procédure de décision.

Durant son stage de DEA, J.-C. Filliâtre a étudié cette procédure de décision et l'a étendue aux cas des formules non prénexes. Il a implanté une version intuitionniste de cette procédure de décision dans le système Coq, sous la forme d'une tactique appelée `Linear` (en raison du lien étroit entre Calcul des Prédicats Direct et Logique Linéaire Multiplicative).

3.2.5 Une interface à Spike

Participant : Judicaël Courant

L'utilisation de techniques de récurrence implicite est un moyen intéressant de prouver automatiquement certains lemmes du premier ordre. Durant son stage de DEA, Judicaël Courant s'est intéressé à une procédure générique de démonstration automatique à base de réécriture proposée par A. Bouhoula dans sa thèse, implémentée dans son prouveur SPIKE. Il a étudié une méthode permettant d'extraire une preuve explicite à partir de la trace de cette procédure générique, et a écrit une implémentation en Caml Light, qui engendre une preuve pour Coq.

3.2.6 Un nouveau système de vérification de preuves

Participants : Gilles Dowek, Hugo Herbelin

G. Dowek et H. Herbelin ont continué le développement (débuté en 1993) d'un prototype d'un nouveau système de traitement de preuves. Ce système combine les idées des deux types classiques de systèmes de traitement de preuves : ceux basés sur la notion de tactique (LCF, HOL, NuPrl, Coq, etc.) et ceux basés sur une méthode de démonstration automatique (Boyer-Moore, Constructor, etc.).

Dans les systèmes basés sur la notion de tactique, l'utilisateur dispose de commandes opérant sur un ensemble de conjectures, le but de l'utilisateur étant de transformer à l'aide de ces commandes sa conjecture en l'ensemble vide de conjectures. Dans ces systèmes, les méthodes de démonstration automatique joue le rôle secondaire de tactiques évoluées. Dans les systèmes basés sur un système de démonstration automatique, en revanche, le système recherche les preuves des conjectures de manière autonome et le rôle de l'utilisateur est limité à la suggestion de lemmes intermédiaires.

Ce nouveau système est également organisé autour d'une méthode de démonstration automatique, qui n'est elle-même qu'un algorithme d'unification d'ordre supérieur généralisé, qui consiste lui-même en l'application successive de substitutions élémentaires. L'utilisateur peut ainsi choisir de laisser le système rechercher les preuves des conjectures en mode automatique ou proposer une substitution élémentaire. Une tactique au sens des systèmes basés sur la première approche est une simple substitution élémentaire de l'algorithme d'unification.

Le noyau très réduit de ce système, lui donne une grande extensibilité et garantit la correction de ces extentions. Tous les exemples distribués avec le système Coq ont été adaptés à ce système. La fécondité de ce genre de structuration d'un noyau de machine de preuve est en cours d'évaluation, notamment en regard de la version courante du système qui encourage une implémentation moins uniforme mais plus modulaire.

3.2.7 Preuves de programmes

Participant : Catherine Parent

Les activités de Catherine Parent concernent la synthèse de preuves à partir de programmes. Il s'agit pour un programmeur averti, souhaitant développer un programme certifié correct, d'obtenir une preuve totalement formelle d'un algorithme pour peu que celui-ci soit donné accompagné de sa spécification ainsi que d'une justification informelle des raisons pour lesquelles l'algorithme est correct.

Pour cela, on se base sur l'idée inverse de l'extraction de programmes à partir de preuves. On part de l'idée que le programme fourni pourrait être extrait de la preuve que l'on cherche. Il est donc le squelette de la preuve recherchée. Ceci permet de retrouver les principales étapes de la preuve, à savoir ses parties calculatoires. Pour les parties non calculatoires, l'aide du programmeur est nécessaire. A savoir, le système va engendrer automatiquement une suite de propriétés logiques à prouver sur le programme, moyennant quoi celui-ci sera vérifié. Le programmeur garde donc à sa charge les justifications de ces propriétés logiques. Cette étude correspond à l'intégration de la tactique `Program` à la version de distribution de Coq.

Les deux difficultés essentielles de ce travail sont d'une part de retrouver les informations concernant les propriétés vérifiées par certaines parties du programme (cela peut correspondre par exemple à un invariant de

boucle), d'autre part, les lourdeurs syntaxiques des programmes extraits inhérentes à la structure de preuve. Pour cela, la possibilité d'introduire une sous-spécification sous la forme d'une annotation a d'une part été ajoutée au langage de description des programmes. D'autre part, des méthodes ont été développées pour pallier les lourdeurs syntaxiques.

Le travail essentiel de cette année a été réparti en deux points. D'une part, l'intégration de la tactique `Program` dans la version V5.10, d'autre part, un travail de formalisation de la tactique. En ce qui concerne la V5.10, la tactique `Program` est en cours de portage sur la structure de types mutuellement inductifs de C.Paulin. Du point de vue formalisation, il s'agit d'expliciter clairement le langage des programmes annotés et de donner une justification théorique de la tactique. Pour cela, une nouvelle notion d'extraction a été définie. Cette extraction définit un langage de programmes annotés et est inversible. Montrer l'inversion de cette extraction revient à donner une méthode explicite de synthèse de preuves à partir de programmes. Cette méthode est correcte et complète et formalise la tactique `Program`. Ce travail est en cours de rédaction dans la thèse de C. Parent qui va être soutenue à l'ENS Lyon courant janvier.

3.3 Axiomatisations Coq

Une part importante de notre recherche consiste à investiguer la représentation des connaissances mathématiques en théorie des types. A cette fin, de nombreuses études de cas ont été menées.

3.3.1 Théorie des Catégories

Participants : Gérard Huet, Amokrane Saibi

G. Huet et A. Saïbi ont commencé un travail de formalisation constructive de la théorie des catégories dans Coq, en utilisant la première partie du développement d'algèbre constructive initié par P. Aczel dans le cadre d'un projet de mécanisation de la théorie de Galois. Ils ont axiomatisé les catégories, les foncteurs, les transformations naturelles, la catégorie SET des ensembles (Setoïds), la catégorie des foncteurs et la famille CAT des catégories de catégories. La loi d'Echange, une propriété non triviale portant sur la commutation entre les compositions verticale et horizontale des transformations naturelles, a été complètement vérifiée dans

Coq. Le lemme de Yoneda, un résultat classique sur la représentation des catégories, a lui aussi été vérifié dans Coq.

Ce développement, réalisé dans la nouvelle version de Coq, a permis de tester ses nouvelles possibilités, notamment l'extension de la syntaxe concrète. La synthèse de type a aussi été très sollicitée : typiquement, lorsqu'une notion ou un lemme est paramétré par deux objets A et B et un morphisme $f \in Hom(A, B)$, nous n'avons besoin de faire explicitement référence qu'à f. Cette facilité est cruciale pour que ce développement corresponde aux notations usuelles en théorie des catégories.

Ce développement est aussi intéressant comme exemple pour comparer les différents systèmes d'aide à la démonstration. En effet des développemets similaires ont été réalisés dans LEGO par P. Aczel et dans ALF par P. Dybjer et V. Gaspes. Un colloque sur l'axiomatisation de la théorie des catégories en théorie des types est prévu pour l'année prochaine.

A. Saïbi étudie maintenant l'axiomatisation de la notion d'adjonction, en vue d'étudier systématiquement en Coq des constructions mathématiques telles que l'initialité.

3.3.2 Arithmétique

Participants : Valérie Ménissier-Morain, Daniel Hirschkoff

Pour axiomatiser le corps des réels en tant que complétion du corps des fractions de $\mathbb{Z}$ en Coq, D. Hirschkoff a implémenté avec V. Ménissier-Morain la tactique Coq *Arith*. Ce travail est basé sur une procédure de décision arithmétique pour les entiers écrite en Caml Light, et sur une spécification des anneaux en Coq ($\mathbb{Z}$ en particulier), toutes deux implémentées par V. Ménissier-Morain. Cette tactique gère des buts arithmétiques simples et décide si ces formules sont satisfaites pour toutes les instanciations de leurs variables. L'ajout principal de D. Hirschkoff au code Caml Light de V. Ménissier-Morain est la production d'arbres de preuve Coq et l'intégration de la tactique au système [33].

Valérie Ménissier a par ailleurs achevé la rédaction de sa thèse sur l'arithmétique exacte [2] et publié un article sur le sujet [16]. Pour plus de précisions voir le chapitre sur le projet Cristal.

3.3.3 Autres études de cas

Participant : Gérard Huet

De nombreux développements ont été effectués en dehors de notre projet. Les développements communiqués par ces utilisateurs sont intégrés progressivement dans une bibliothèque de théories disponibles pour tous dans une zone CONTRIB du système distribué. On peut citer les développements récents suivants:

- L'algorithme d'unification de 1er-ordre de Robinson a été prouvé correct par J. Rouyer et P. Lescanne, de l'INRIA Lorraine.
- Des algorithmes de manipulation d'arbres de Shannon, formant le cœur des algorithmes BDD de vérification de logique Booléenne ont été spécifiés et vérifiés par E. Ledinot de la société Dassault Aviation.
- La représentation de notions sémantiques en Coq a été étudiée par J. Despeyroux de l'INRIA-Sophia, et P. Casteran du LABRI à Bordeaux.
- Un traducteur Coq-Typol a été implémenté par D. Terrasse de l'INRIA-Sophia.
- Une interface Coq-Centaur (CTCoq) a été implantée par Y. Bertot de l'INRIA-Sophia.
- La linguistique des preuves formelles issues de Coq a été étudiée, et un imprimeur de preuves en LaTeX lisible a été implémenté par A. Massol de l'INRIA-Sophia.
- La formalisation de machines virtuelles B en Coq a été étudiée par R. Fraer, dans le cadre d'un stage de maîtrise de l'ENS Lyon effectué à Alsthom.
- La formalisation des notions de base de la théorie des langages formels et des automates a été étudiée par J. Courant et J.P. Filliâtre dans le cadre d'un stage de maîtrise de l'ENS Paris.
- La formalisation des "process algebras" et du langage μCRL en Coq a été étudiée par L. Helmink de la société Philips à Eindhoven, M. Bezem et M. P. Sellink de l'Université d'Utrecht.
- Cette formalisation a été utilisée par Jan Friso Groote de l'Université d'Utrecht pour prouver la correction du protocole du bit

alterné et celle du protocole "bounded resources" utilisé par la société Philips. Cette dernière preuve est la plus considérable réalisée à ce jour dans notre système.

- Le théorème de Ramsey en dimension 2 a été démontré en Coq par M. Bezem de l'Université d'Utrecht.
- Le théorème de Rem dans les espaces de Baire a été démontré en Coq par H. Barendregt de l'Université de Nijmegen.
- La théorie des treillis continus a été étudiée par G. Kahn et F. Prost de l'INRIA-Sophia.

4 Actions contractuelles

Nous participons au programme ESPRIT BRA de recherche fondamentale. Une action, centrée sur le thème "Types", rassemble le projet Coq, l'Université d'Edimbourg, l'Université de Cambridge, le projet CROAP de l'INRIA Sophia-Antipolis, le groupe de Logique Mathématique de l'Université Paris 7, l'Université de Manchester, le TUM de Munich, la société Philips à Eindhoven, l'Université de Nijmegen, l'Université de Turin, et l'Université de Göteborg, comme partenaires. L'INRIA est contractant principal, G. Huet est coordinateur de l'action.

Nous participons à une opération Inter-PRC appelée "Mécanisation du Raisonnement". Les principaux sites contractants sont l'Inria-Lorraine-CRIN, le LRI à l'université Paris-Sud, l'IRIT à Toulouse et le LIFIA-IMAG à Grenoble.

G. Huet a réalisé pour le SGDN une étude sur la certification du logiciel et les méthodes formelles en génie logiciel. Il a présenté le rapport correspondant[34] au SGDN, à des séminaires chez Bull et Thomson, et devant l'Observatoire de la recherche en Informatique. Il a également fait une conférence sur l'utilisation des méthodes formelles en génie logiciel à l'Ecole Polytechnique en Octobre. Cette action se poursuit par la participation à un groupe de travail de l'OFTA (Observatoire Français des techniques avancées) sur la certification du logiciel qui vient de se former pour 2 ans. G. Huet y représente l'INRIA.

Dans le cadre de l'appel d'offres MRT 1992, nous avons conclu avec l'ENS Paris et la société Dassault une convention sur le thème "Ingénierie Logicielle pour le développement de programmes temps-réel sûrs." Cette convention s'étend sur 2 ans, à partir de Novembre 92, elle vient

de se terminer. Un bilan de la problématique utilisée sera effectué lors d'une journée de travail en janvier 1995.

Nous participons au projet Génie, commun à l'INRIA et à Dassault, qui est en phase de démarrage. Par ailleurs, nous sommes en négociation pour une collaboration avec l'équipe de D. Bolignano du centre de recherches Bull sur la preuve de protocoles de communication.

5 Diffusion des résultats

Les résultats obtenus par l'équipe sont largement diffusés dans des colloques nationaux et internationaux et dans des publications. Le séminaire Coq-Cristal-Para, organisé par D. Rémy, a accueilli cette année une trentaine de participants internationaux. Nous continuons bien sûr à avoir des rapports privilégiés avec le projet Cristal.

Nous formons une équipe du GDR Programmation. C. Paulin est responsable du pôle "Preuves et Spécifications Algébriques" de ce GDR. Dans le cadre de l'action "Mécanisation du Raisonnement", l'équipe a participé à un colloque de 2 jours à Chamrousse en novembre.

Les projets du GDR sont donc des partenaires privilégiés. Citons des rapports fréquents avec le groupe de J.P. Jouannaud à l'Université d'Orsay sur les problèmes de réécriture, le projet Croap de l'INRIA-Sophia sur les problèmes de sémantique opérationnelle et d'interfaces de systèmes de preuves, le projet Para de l'INRIA-Rocquencourt pour les problèmes de λ-calcul et réécriture.

A l'étranger, nous sommes en contact régulier avec nos partenaires européens BRA, et avec les universités Carnegie-Mellon, U. de Pennsylvanie, U. Stanford, Centre DEC-SRC, AIT Bangkok, IIT Kanpur, etc.

Le colloque annuel du BRA "Types", organisé en Mai à Bastad, a réuni une centaine de participants. Un volume de Proceedings de 300 pages a été édité électroniquement. Un livre rassemblant une sélection des contributions est en cours d'édition. Le livre correspondant au workshop de 1993, édité par H.Barendregt et T. Nipkow, est paru cette année chez Springer-Verlag.

5.1 Colloques et congrès

B. Werner a participé au neuvième symposium LICS à Paris où il a presenté un travail commun avec Herman Geuvers[21].

C. Paulin, C. Parent, C. Cornes, S. Boutin, G. Huet, A. Saïbi et B. Werner ont participé au workshop du BRA *Types* à Båstad.

C. Paulin a participé à la conférence LPAR'94 à Kiev où elle a présenté unintroduction à Coq. C. Paulin a été invitée à faire une présentation au workshop "Logic and Computer Science around the 42th //" à Luminy où elle a présenté son travail sur les "streams" et les circuits.

G. Dowek et G. Huet ont participé à la conférence CADE à Nancy en Juin. G. Huet y a participé à une table ronde "QED" organisée par B. Boyer, avec la participation de N. de Bruijn et de A. Trybulec.

G. Huet a participé au Symposium Automath à Eindhoven en Septembre. Il y a présenté une conférence invitée sur "Proof engine design". Il a participé en Mai à la review de la société CLINK à Austin au Texas, où il a présenté les travaux du projet Coq.

C. Parent et J. Courant ont participé aux journées du GDR Programmation en septembre à Lille. J. Courant a présenté un exposé sur l'explicitation de preuves implicites.

E. Giménez, C. Muñoz et A. Saïbi ont participé à l'École de Jeunes Chercheurs du Greco de Programmation à Toulouse en Mai 1994, où G. Huet a enseigné un cours de λ-calcul.

C. Muñoz a participé au colloque *Logique et Informatique* à Marseille en Juin 1994.

C. Cornes a participé à l'école d'été *Logic for Computer Science on Automated Deduction* en Juillet 1994 à Chambery.

E. Giménez et J.-C. Filliâtre ont participé à l'école d'été *Advanced School on Typed Lambda Calculus and Fuctional Programming* en Septembre 1994 à Udine (Italie).

P. Audebaud, H. Herbelin et P. Manoury ont participé au workshop "Computational interpretations of classical proofs" du BRA organisé en décembre 93 à Marseille-Luminy.

H. Herbelin a participé à la conférence Computer Science Logic qui s'est tenue à Kazimierz (Pologne) en septembre 1994. Il y a présenté une

communication sur une structure de λ-calcul isomorphe à la structure des calculs de séquents du type des calculs LJ et LK de Gentzen.

Le projet a largement participé aux journées du GDR Programmation à Toulouse en Février et à Grenoble en Novembre.

5.2 Responsabilités éditoriales et professionnelles

G. Huet est membre de "Academia Europaea" et membre correspondant de l'Académie des Sciences. Il est membre élu de la Commission d'Evaluation de l'INRIA, et à ce titre a participé cette année aux séminaires d'évaluation du programme 3 à Grenoble en Février et du programme 2 à Jouy en Septembre, ainsi qu'à une journée d'expertise à Sophia en octobre. Il a également participé au colloque thématique sur l'avenir de la recherche à Bordeaux en Mars.

G. Huet fait partie des comités de lecture des revues "Journal of Symbolic Computation", "Journal of Applied Non-Classical Logics", "Journal of Logic and Computation", "International Journal on Foundations of Computer Science", "Annals of Pure and Applied Logic", et de la "Revue d'Intelligence Artificielle". G. Huet est membre du Comité Scientifique de l'IRISA, ainsi que de celui du LIFL-URA 369 (Lille) et de celui du LIP (Lyon). Il est membre du Conseil Scientifique du PRC "Maths-Info" et du "Honorary Board" du RIDS (Research Institute for Declarative Systems) in Nijmegen. Il est rapporteur du projet ESPRIT BRA CLICS2; dans ce rôle, il a participé à la revue du projet à Londres en octobre.

C. Murthy a reçu cette année une bourse Chateaubriand, co-financée par le Ministère des Affaires Etrangères et l'INRIA.

C. Paulin a été membre du comité de programme de la conférence "Logic Programming and Automated Reasoning 94".

5.3 Participations à des actions d'enseignement

Les chercheurs de notre équipe, qu'ils soient du CNRS, de l'INRIA ou boursiers, font un effort d'enseignement important, particulièrement en ce qui concerne les cours de troisième cycle.

G. Huet a enseigné 8h à l'Ecole de la recherche du Greco de Programmation à Toulouse en Mai 1994. Il a également enseigné une

semaine à l'école "Advanced School on typed λ-calculus and functional programming", à Udine en Septembre.

G. Huet et G. Dowek enseignent un cours intitulé "Formalisation des Concepts Mathématiques" dans le DEA "Informatique Fondamentale" de l'Université Paris VII.

G. Dowek a donné un cours "Automated deduction in type theory" à la "2nd Chambéry International Summer School on Automated Deduction" organisée par l'Université de Paris 7 et l'Université de Savoie.

B. Werner est chargé de TP vacataire à l'Ecole Polytechnique dans le cours d'initiation à l'algorithmique et à la programmation du tronc commun de première année.

C. Paulin participe au DEA d'Informatique Fondamentale de l'Ecole Normale Supérieure de Lyon (cours de 20h intitulé Preuves et programmes, année 93-94).

G. Huet encadre le travail de thèse de S. Boutin, de C. Cornes, de V. Ménissier, d'A. Saïbi et de C. Muñoz; C. Paulin encadre le travail de thèse de C. Parent, d'E. Gimenez et de J.-C. Filliâtre. Il a participé aux soutenances de thèse de B. Monsuez à l'Ecole Polytechnique en février, de C. Raffali à Paris VII en février, de A. Bouhoula et J. Rouyer à Nancy en mars, de B. Werner à Paris VII en mai, d'E. Poll à Eindhoven en septembre, de V. Ménissier à Paris VII en décembre. La soutenance des thèses de H. Herbelin à Paris VII et de C. Parent à Lyon est prévue pour janvier 1995.

Les activités d'enseignement de P. Audebaud se font dans le cadre de l'Ecole Normale Supérieure de Lyon. Il y assure les travaux dirigés des modules suivants de deuxième année du Magistère d'Informatique : programmation logique, programmation fonctionnelle et orientée objets, et sémantique des langages de programmation.

Attaché Temporaire d'Enseignement et de Recherche (ATER) pour l'année scolaire 1994/95, H. Herbelin a assuré des Travaux Dirigés sur le langage C, sur UNIX et sur l'architecture des ordinateurs, aux élèves de licence et d'informatique de l'université Paris 7.

Claude Marché est Attaché Temporaire d'Enseignement et de Recherche à l'École Normale Supérieure de Cachan depuis le premier septembre 1994, où il y assure des cours de programmation (Fortran, C) et d'algorithmique dans diverses sections de l'ENS (Génie électrique, Mécanique,

Biochimie, Économie). Il continue en même temps le travail de recherche commencé pendant la bourse post-doc.

C. Parent assure des Travaux Dirigés de Sémantique en 2ème année de Magistére d'Informatique et Modélisation de l'Ecole Normale Supérieure de Lyon

Valérie Ménissier-Morain a assuré, pendant le second semestre de l'année universitaire 1993-1994, des TD/TP de Maple en DEUG MIAS à l'Université de Versailles Saint-Quentin en Yvelines. Elle occupe depuis septembre 1994 un demi-poste d'ATER à l'Université d'Évry Val d'essonne où elle enseigne l'algorithmique à travers Caml en licence de mathématiques et la programmation fonctionnelle avec Caml en seconde année de MIAGE.

S. Boutin enseigne en premier cycle au Conservatoire National des Arts et Métiers depuis Octobre 93.

6 Publications

Thèses

[1] H. Herbelin, *Elimination des coupures et l'interprétation théorie des jeux en calcul des séquents classique*, thèse de doctorat, Université Paris VII, Déc. 1994.

[2] V. Ménissier-Morain, *Arithmétique exacte. Conception, algorithmique et performances d'une implémentation informatique en précision arbitraire*, thèse de doctorat, Université Paris VII, Déc. 1994.

[3] B. Werner, *Une Théorie des Constructions Inductives*, thèse de doctorat, Université Paris VII, Mai. 1994.

Articles et chapitres de livre

[4] P. Audebaud, «Explicit Substitutions for the Lambda-Mu Calculus», *Submitted*, 1994.

[5] T. Coquand, H. Herbelin, «A-translation and Looping Combinators in Type Systems», *Journal of Functional Programming 4*, 1994, p. 77–88.

[6] T. Coquand, H. Herbelin, «Some remarks on Novikoff's calculus», *Manuscript*, 1994.

[7] G. Dowek, «Third order matching is decidable», *Annals of Pure and Applied Logic, 69, pp. 135-155*, 1994.

[8] E. Giménez, «Codifying guarded definitions with recursive schemes», *In preparation*, 1994.

[9] H. Herbelin, «ESPRIT Basic Research Action 6453, TYPES: Types for Proofs and Programs», *To appear in the Bulletin of the European Association for Theoretical Computer Science*, 1994.

[10] H. Herbelin, «Lorenzen's games and Gentzen-like sequent calculi», *In preparation*, 1994.

[11] H. Herbelin, «Weak cut-elimination and the game-theoretic approach of provability», *Manuscript*, 1994.

[12] G. Huet, A. Saïbi, «Constructive Category Theory», *In preparation*, 1994.

[13] G. Huet, «An analysis of Böhm's theorem», *in : To C. Böhm: Essays on Lambda-Calculus and Functional Programming*, S. R. d. R. M. Dezani-Ciancaglini et M. V. Zilli (réd.), Cambridge University Press, 1993.

[14] G. Huet, «Residual theory in λ-calculus: a formal development"», *J. of Functional Programming 4,3*, 1994, p. 371–394.

[15] P. Manoury, M. Simonot, «Automatizing termination proof of recursively defined function», *TCS*, To appear.

[16] V. Ménissier-Morain, P. Weis, «An exact arithmetic package for ML», *Science for Computer Programming*, 1994, À paraître.

[17] C. Paulin-Mohring, B. Werner, «Synthesis of ML programs in the system Coq», *Journal of Symbolic Computation 15*, 1993, p. 607–640.

Communications à des congrès, colloques, etc.

[18] G. Dowek, T. Hardin, C. Kirchner, «Higher Order Unification via Explicit Substitutions», *in : Submitted*, 1994.

[19] G. Dowek, «Automated theorem proving in type theory, course notes», *in : Second international summer school in logic for computer science, Chambery France*, M. Parigot (réd.), 1994.

[20] G. Dowek, «Lambda-calculus, Combinators and the Comprehension Schema», *in : Proceedings of the second international conference on typed lambda calculus and applications*, 1995.

[21] H. Geuvers, B. Werner, «On the Church-Rosser Property for Expressive Type Systems and its Consequences for their Metatheoretic Study», *in : Proceedings of LICS 94, Paris, France*, S. Abramsky (réd.), 1994.

[22] H. Herbelin, «A λ-calculus structure isomorphic to sequent calculus structure», *in : Proceedings of CSL 94, Kazimierz, Poland*, 1994.

[23] F. Leclerc, C. Paulin-Mohring, « Programming with Streams in Coq. A case study: the Sieve of Eratosthenes », *in : Proceedings of Types'93*, H. Barendregt, T. Nipkow (réd.), 1994.

[24] C. Marché, « Associative-Commutative Reduction Orderings via Head-Preserving Interpretations », Submitted to RTA'95.

[25] C. Marché, « Normalized rewriting and normalized completion », *in : Proceedings of LICS 94, Paris, France*, S. Abramsky (réd.), 1994.

[26] C. Parent, « Developing certified programs in the system Coq - The Program tactic », *in : Proceedings of Types'93*, H. Barendregt, T. Nipkow (réd.), 1994. Also available as a report LIP-ENS Lyon RR 93-29 and by anonymous ftp from lip.ens-lyon.fr file pub/Rapports/RR/RR93/RR93-29.ps.Z.

[27] C. Paulin-Mohring, « Inductive Definitions in the System Coq - Rules and Properties », *in : Proceedings of the conference Typed Lambda Calculi and Applications*, M. Bezem, J.-F. Groote (réd.), *LNCS*, 664, 1993.

Rapports de recherche et publications internes

[28] C. Barraband, *Etude du principe d'induction généralisée pour les types de données inductifs*, Mémoire, DEA d'Informatique Fondamentale, Université Paris 7, Septembre 1994.

[29] J. Courant, *Explicitation de preuves par récurrence implicite*, Mémoire, DEA d'Informatique, ENS Lyon, Septembre 1994.

[30] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Murthy, C. Parent, C. Paulin-Mohring, B. Werner, « The Coq Proof Assistant User's Guide. Version 5.8 », *rapport de recherche n° 154*, INRIA, Mai 1993.

[31] G. Dowek, « Collections, Sets and Types », *rapport de recherche*, 1994.

[32] J.-C. Filliâtre, *Une procédure de décision pour le Calcul des Prédicats Direct. Etude et implémentation dans le système Coq*, Mémoire, DEA d'Informatique, ENS Lyon, Septembre 1994.

[33] D. Hirschkoff, *Ecriture d'une tactique arithmétique pour le système Coq*, Mémoire, DEA IARFA, Ecole des Ponts et Chaussées, Paris, Septembre 1994.

[34] G. Huet, « Certification du logiciel: Méthodes et outils. Etat de l'art des méthodes formelles en génie logiciel », *rapport de recherche n° 11/SGDN/STS/VST*, SGDN, April 1994.

[35] G. Huet, « The Coq Tutorial », *rapport de recherche n° En préparation*, INRIA, 1994.

[36] C. Munoz, *Démonstration automatique dans la logique propositionnelle intuitionniste*, Mémoire, DEA d'Informatique Fondamentale, Université Paris 7, Septembre 1994.

[37] C. Parent, «Developing certified programs in the system Coq- The Program tactic», *rapport de recherche n° 93-29*, Ecole Normale Supérieure de Lyon, octobre 1993, also in Informal Proceedings of the BRA Workshop n Types for Proofs and Programs, may 1993.

[38] C. Paulin-Mohring, «Définitions Inductives en Théorie des Types d'Ordre Supérieur», Mémoire d'habilitation en préparation, 1995.

[39] A. Saïbi, «Axiomatization of a lambda-calculus with explicit-substitutions in the Coq System», *rapport de recherche n° 2345*, INRIA, Dec 1994.

[40] H. Saidi, *Unification dans le système T*, Mémoire, DEA d'Informatique Fondamentale, Université Paris 7, Septembre 1994.

7 Abstract

The Coq research team is interested in the design and implementation of proof assistants well suited for the development of certified software, and more generally to the research area of type theory.

Table des matières