Projet COQ

Spécifications et preuves de programmes

Localisation : *Rocquencourt*

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

1 Composition de l'équipe

Responsables Scientifiques

Gérard Huet, Directeur de Recherche INRIA
Christine Paulin, Chargée de Recherche CNRS, ENS-Lyon

Responsable Permanent

Gilles Dowek, Chargé de Recherche INRIA

Secrétaire (commun avec les projets Cristal et Para)

Sylvie Loubressac, INRIA

Chercheurs confirmés

Philippe Audebaud, Maître de Conférences, ENS-Lyon
Jean Duprat, Maître de Conférences, ENS-Lyon
Jean Goubault-Larrecq, Chercheur Dyade depuis 09/96
Hugo Herbelin, ATER, Paris 7/Maître de Conférences, Paris 10
Benjamin Werner, Chargé de Recherche INRIA

Chercheurs de 3ème cycle

Bruno Barras, bourse MENESR
Samuel Boutin, moniteur normalien jusqu'en 09/96
Cristina Cornes, bourse INRIA-Génie
Judicaël Courant, normalien ENS-Paris/moniteur normalien
Jean-Christophe Filliâtre, normalien ENS-Paris, Service National jusqu'en 07/96
Eduardo Giménez, bourse INRIA-Génie
Henri Laulhère, bourse MENESR
César Muñoz, bourse CROUS
Stéphane Philipakis, bourse Dyade jusqu'en 09/96
Frédéric Prost, bourse MENESR-Génie
Amokrane Saïbi, bourse gvt. algérien/ATER, Paris 6

2 Présentation du projet

Le projet Coq a pour but la réalisation de *systèmes de traitement de démonstrations*, c'est-à-dire de systèmes capables de vérifier, produire et transformer des démonstrations mathématiques.

Ces systèmes peuvent en particulier être utilisés pour vérifier les démonstrations de correction de matériels et de logiciels informatiques vis-à-vis d'une spécification formelle, permettant ainsi la production de produits informatiques de qualité totale (*zéro défaut*). Ils construisent explicitement un objet représentant la preuve de correction du programme qui peut ainsi être à nouveau vérifiée dans le cadre de la certification d'un logiciel.

Dans la conception d'un système de traitement de démonstrations, le choix du formalisme dans lequel s'expriment les définitions, les axiomes, les théorèmes et les démonstrations est crucial. La théorie des ensembles n'est malheureusement pas totalement adaptée aux buts orientés vers l'informatique que nous poursuivons. Nous travaillons donc avec une variante de la théorie des ensembles appelée *Le Calcul des Constructions Inductives*.

La principale originalité de ce formalisme est que les démonstrations y sont des objets au même titre que les nombres, les fonctions ou les ensembles. Ainsi un entier pair sera représenté par un couple formé d'un entier et d'une preuve que cet entier est pair. La seconde originalité de ce formalisme est que chaque terme exprimant un objet d'un type de données, peut se réduire sur une valeur de ce type de données. Par exemple le terme $2 + 5$ se réduit sur la valeur 7.

Ces propriétés mises ensemble permettent de concevoir la spécification d'un programme comme une relation liant la valeur d'entrée et la valeur de sortie de ce programme. En effet, si $Q(x, y)$ décrit une telle relation, une preuve dans le *Calcul des Constructions Inductives* de la totalité de cette relation $(\forall x.\exists y.Q(x, y))$ est en fait une fonction qui, à tout objet a associe un objet b et une preuve de $Q(a, b)$. À partir de cette preuve, il est possible d'exprimer dans le calcul à la fois un programme fonctionnel f et sa preuve de correction $\forall x.Q(x, f(x))$. La fonction f appliquée à l'entrée a se réduira pour fournir la sortie b.

Mais les preuves ne sont pas, en général, des programmes efficaces, et il est nécessaire, en pratique, d'éliminer certaines parties de ces preuves non pertinentes pour le calcul : cette étape s'appelle l'extraction de programme. Les programmes extraits peuvent alors être traduits dans un langage de programmation ordinaire tel que ML, puis compilés en code machine.

Dans cette approche, la spécification du programme est vue comme une formule mathématique et le programme certifié est identifié à la preuve de cette formule.

Le langage de spécification associé au *Calcul des Constructions Inductives* (appelé Gallina) contient en particulier un calcul des prédicats d'ordre supérieur typé permettant d'exprimer des propriétés sur des objets appartenant à des structures algébriques quelconques. Il permet une représentation directe sans codage excessif des notions fondamentales d'objets structurés, de fonctions ou de relations très souvent utilisées dans les preuves informatiques.

La puissance même du système logique rend *a priori* impossible l'automatisation complète de la recherche d'une preuve d'une formule. L'approche repose sur deux autres idées. La première est qu'étant donnée une démonstration formelle assez détaillée, il est possible de vérifier mécaniquement sa correction. La seconde est que des programmes arbitraires (appelés tactiques) peuvent être utilisés pour construire par étapes successives une démonstration dont la correction est ensuite vérifiée. Ceci permet de laisser l'utilisateur construire ses propres procédures tout en garantissant la correction des preuves finales.

Une partie de notre activité concerne le développement et l'expérimentation d'un système de traitement des démonstrations : le système Coq qui est fondé sur le *Calcul des Constructions Inductives*. Nous nous intéressons à deux grandes classes de problèmes pouvant être abordés avec cet outil. Le premier est la certification de code mobile sur le réseau Internet et son application au télé-paiement dans le cadre du projet VIP avec le GIE Dyade. Le second est la certification d'outils automatiques tels que des compilateurs mis en œuvre dans des développements de systèmes réactifs critiques.

Parallèlement, nous menons des activités de recherche à plus long terme concernant l'extraction, et plus généralement, la preuve de programmes, l'architecture des systèmes de vérification de démonstrations (en particulier la recherche d'une architecture suffisamment simple pour être elle-même spécifiée et vérifiée), la modélisation de structures fondamentales des objets de preuve et l'étude du *Calcul des Constructions Inductives* ainsi que certaines de ses extensions et alternatives possibles.

3 Actions de recherche

3.1 Le développement du système Coq

3.1.1 Fonctionnalités de Coq

Coq est un assistant à la démonstration. Ses principales fonctionnalités sont au niveau du formalisme :

- Une structure primitive de définitions mutuellement inductives permettant des spécifications de haut niveau soit dans un style fonctionnel en déclarant des types concrets et en définissant des fonctions par des équations représentant un calcul, soit dans un style déclaratif en spécifiant des relations à l'aide de clauses.
- Une structure primitive de définitions co-inductives permettant de représenter directement des structures infinies régulières et de construire des preuves sur de tels objets sans passer par la notion classique de bi-simulation.
- Une interprétation des preuves comme des programmes certifiés mise en œuvre dans une compilation des preuves sous forme de programmes ML mais aussi des outils pour associer un programme à une spécification et engendrer automatiquement des obligations de preuve permettant de justifier sa correction.

Au niveau de l'architecture de l'assistant :

- Un système de bibliothèques mathématiques modulaires permettant de compiler et recharger rapidement des théories mathématiques.
- La possibilité d'introduire des notations spécifiques par l'utilisation de grammaires et fonctions d'impression interprétées.
- La possibilité de développer des tactiques comme des programmes ML sophistiqués qui peuvent ensuite être chargés et utilisés dans l'environnement.

3.1.2 Diffusion

Le système Coq a une communauté significative d'utilisateurs et de contributeurs. Nous avons décompté plus de 60 téléchargements du système au mois de février lors de la nouvelle livraison. Outre les deux sites de notre projet, nous entretenons des collaborations avec d'autres groupes de recherches qui collaborent à notre effort. Au premier rang de ceux-ci, figurent nos collègues du projet Croap de Sophia-Antipolis qui développent en Centaur, une interface très sophistiquée baptisée *CtCoq* et qui travaillent également sur la traduction des preuves formelles en langue naturelle et la formalisation de preuves de sémantique des langages. Nous avons également des liens très étroits avec des équipes de Paris 6 (Th. Hardin) et du CNAM (V. Donzeau-Gouge), qui organisent un séminaire consacré aux systèmes Coq et *B*. Nous collaborons également avec des équipes de Nancy, de Marseille et de Bordeaux.

Nos utilisateurs industriels au CNET (P. Crégut et J.-F. Monin), chez Dassault (E. Ledinot) et chez Bull (D. Bolignano) contribuent également à notre bibliothèque de démonstrations.

Le manuel de référence [47] et un manuel introductif sont régulièrement mis à jour et ont été complétés de documentations spécifiques.

Depuis sa diffusion, le système Coq a été installé sur 250 sites environ et la liste de courrier Coq-Club concerne 120 utilisateurs réguliers pour moitié en France et pour moitié dans le reste du monde.

C. Paulin a organisé une formation à Coq et CtCoq de trois jours en mars qui a été suivie par 14 personnes (universitaires et industriels). Y. Bertot, G. Huet, G. Kahn et C. Paulin sont intervenus dans cette formation et E. Giménez a rédigé à cette occasion une introduction aux types récursifs [56]. Plusieurs demandes de participation n'ont pu être satisfaites faute de place.

3.1.3 Coq V5.10 et V6.1

La dernière distribution de Coq V5.10 a été réalisée en février. H. Laulhère a porté cette version dans les environnements Windows 95 et Windows NT.

Cette année, nous avons développé la version Coq V6.1 dont la première distribution en alpha-test a été fournie à nos utilisateurs les plus proches en octobre. Une version distribuée sur Internet, en FTP devrait être disponible fin décembre.

Les principaux changements intervenus dans le système Coq V6.1 sont les suivants:

- Le portage du code de Coq en Objective Caml effectué par B. Barras. Il est désormais possible de compiler Coq en code natif, ce qui apporte un gain conséquent en rapidité.
- Des mécanismes de syntaxe implicite réalisés par A. Saïbi. Ils permettent de simplifier la spécification des problèmes en prenant en compte de manière automatique les "raccourcis" habituels du langage mathématique.
- Un langage riche de motifs pour exprimer l'analyse par cas suivant les constructeurs d'un type concret réalisé par C. Cornes.
- Des tactiques génériques pour la réécriture réalisées par A. Saïbi.
- De nouvelles tactiques :
  - S. Boutin a écrit et intégré une procédure de décision pour la théorie des anneaux abéliens
  - H. Herbelin a intégré la tactique `Omega` réalisée par P. Crégut du CNET qui implémente une procédure de décision partielle sur un sous-ensemble de l'arithmétique de Presburger.
  - C. Paulin a réalisé une tactique de résolution à la Prolog prenant en compte une liste de *hints*.
- Affichage des preuves : H. Herbelin a intégré l'outil d'affichage des preuves en langue naturelle de Y. Coscoy du projet Croap.
- Extraction de programmes : J.-C. Filliâtre a ajouté un module d'extraction des programmes vers le langage *Objective Caml*.
- Nouvelles sortes. Les sortes servent à classifier les objets pour éviter les paradoxes. C. Paulin a simplifié la structure des sortes pour permettre d'uniformiser les bibliothèques de base.

3.1.4 Coercions Implicites

Participant : Amokrane Saïbi

A. Saïbi a défini et implémenté dans Coq un mécanisme d'héritage [43] visant à améliorer la syntaxe des termes. Étant donné un terme mal typé, le mécanisme d'héritage détermine s'il peut être bien typé modulo l'insertion de coercions appropriées. La syntaxe est ainsi plus souple; en effet les écritures suivantes sont dorénavant permises:

- $(f\ a)$ où $f : (x : A)B$ et $a : A'$ si A' peut, dans un certain sens, être vu comme un sous-type de A.

– $x : A$ alors que A n'est pas un type, mais peut être considéré comme un type: ensemble, groupe, catégorie etc.

– $(f\ a)$ alors que f n'est pas une fonction, mais peut être vue comme telle: bijection, foncteur, morphisme de groupe etc.

Ce travail constitue une extension de la proposition de Peter Aczel par la prise en compte des classes avec paramètres et la définition de deux classes abstraites `SORTCLASS` (la classe des sortes) et `FUNCLASS` (classe des fonctions).

3.1.5 Réécriture Générique

Participant : Amokrane Saïbi

A. Saïbi a implémenté pour Coq un paquetage de réécriture multi-relations similaire à celui de Nuprl, implémenté par Paul Jackson. L'utilisateur dispose d'un langage de *conversions* et de *conversionals*, dans le même esprit que celui des tactiques et tacticals, et pouvant être combinées de différentes manières, lui permettant de définir ses propres stratégies de réécriture (eventuellement très sophistiquées). Cette technique a été adoptée dans les systèmes HOL, Isabelle et Nuprl.

3.1.6 Définition par filtrage

Participante : Cristina Cornes

C. Cornes a implanté une stratégie pour compiler le filtrage de motifs avec types dépendants en Coq. Ceci permet d'avoir dans ce système une syntaxe pour les programmes plus proche de celle des langages de programmation fonctionnels à la ML.

Ainsi la fonction de soustraction sur les entiers pourra s'écrire directement comme :

```
Fixpoint minus [n:nat] : nat -> nat :=
  [m:nat]Cases n m of
             O     _    =>  O
           | (S k)  O    => (S k)
           | (S k) (S l) => (minus k l)
          end.
```

Cette définition est analysée pour vérifier sa complétude et est compilée vers une expression du formalisme de base qui ne comporte que du filtrage à un seul niveau.

3.2 Utilisation du système Coq

Nous développons dans le système Coq des exemples typiques.

3.2.1 Réflexion

Participant : Samuel Boutin

Samuel Boutin a exploré les techniques de réflexion qui consistent à représenter comme un objet la proposition à démontrer. Il a montré qu'on pouvait ainsi définir des procédures de décision dans le système Coq, et surtout, internaliser leur preuve de correction. Il a montré que l'on obtenait ainsi également un gain sensible en efficacité. Samuel Boutin a expérimenté cette approche avec l'exemple de la décision sur les anneaux abéliens.

3.2.2 Théorie des Ensembles en Coq

Participant : Benjamin Werner

Les preuves de normalisation et de cohérence du formalisme de Coq semblent nécessiter l'existence de cardinaux inaccessibles. Il était donc intéressant de vérifier si le formalisme de Coq n'était pas logiquement plus puissant que la théorie des ensembles de Zermolo-Fraenkel. Benjamin Werner a développé des travaux de Peter Aczel pour encoder en Coq la théorie ZFC. En ajoutant à Coq un axiome du choix non-calculatoire, il est possible de construire une formalisation complète de ZFC; les axiomes habituels de cette dernière deviennent alors des théorèmes de Coq. Les conséquences principales sont:

- Une preuve immédiate de cohérence relative de ZFC dans Coq muni de l'axiome du choix.
- Une ouverture quand aux possibilités de formaliser en Coq les mathématiques traditionnellles.
- Une vision intéressante des rapports entre formalismes intentionnels et extentionnels, ainsi qu'entre formalisations constructive et non-constructive en Coq.

3.2.3 Formalisation des arbres de Böhm rationnels

Participant : Henri Laulhère

H. Laulhère a commencé une axiomatisation des arbres de Böhm rationnels. Ceci afin de conforter la description théorique du formalisme.

3.2.4 Preuves de circuit

Participant : Jean Duprat

J. Duprat poursuit ses recherches sur les preuves de circuits dans deux directions. Il étudie d'une part, la représentation d'un circuit sous la forme d'un réseau d'automates et d'autre part, la caractérisation d'algorithmes admettant une implantation séquentielle et une implantation parallèle.

3.3 Preuves de programmes

Nous travaillons à comprendre comment un système de traitement de preuve peut être utilisé en génie logiciel pour développer des programmes certifiés. Notre travail porte sur le développement de méthodologies et d'outils permettant d'obtenir des programmes efficaces et de dégager des principes de preuves systématiques.

3.3.1 Récursion bien fondée

Participant : Cristina Cornes

C. Cornes étudie l'automatisation de preuves de terminaison de fonctions récursives définies équationnellement. À partir de la méthode de synthèse de preuves proposée par Gilles Dowek pour les *Systèmes de Types du Cube*, et celle proposé par Hassan Saidi pour le *Système T* de Gödel, C. Cornes a proposé une méthode de synthèse des preuves pour le *Calcul des Constructions Inductives*. La complétude de cette méthode est en cours d'étude et fournirait un cadre général où l'on peut définir des stratégies de recherche de preuves (dont celles pour prouver la terminaison) pour ce calcul. Son travail permettra à terme d'ajouter au système Coq des heuristiques pour prouver la terminaison des fonctions récursives définies par équations et en particulier, d'étendre la commande `Recursive Definition` proposée par Pascal Manoury dans le système Coq.

3.3.2 Traits Impératifs

Participant : Jean-Christophe Filliâtre

Dans le cadre de sa thèse, J.-C. Filliâtre étudie comment prouver des programmes impératifs, plus précisément des programmes écrits dans un langage fonctionnel comportant des aspects impératifs. Un langage fonctionnel comprenant exceptions et références fournit, dans un premier temps, un bon cadre de travail. Une manière de procéder consiste à transformer les programmes impératifs en termes fonctionnels traduisant leur sémantique. La preuve que le programme satisfait une certaine spécification se fait alors sur la traduction du programme et non sur le programme lui-même. Les monades, introduites par E. Moggi, donnent un cadre assez général pour de telles traductions : elles permettent de définir, pour un type de données T, le type des *calculs de type* T. J.-C. Filliâtre a défini de telles traductions et les expérimente sur des programmes de tri en place (quicksort, heapsort).

3.3.3 Efficacité des programmes extraits

Participant : Frédéric Prost

La communauté travaillant dans le cadre de l'extraction s'interroge sur les propriétés à donner aux systèmes en vue d'une extraction optimale. Dans [53], F. Prost avait montré l'intérêt des langages utilisant des termes marqués. Poursuivant son travail sur ce sujet, il est apparu que de nombreuses approches utilisant le sous-typage ou les types conjonctifs ont été proposées pour résoudre un problème apparaissant naturellement dans le cadre de l'extraction de programme: les fonctions surchargées. En utilisant une version modifiée du $\lambda\&$-calcul de G. Castagna, système permettant de traiter des fonctions surchargées, F. Prost a pu donner un cadre permettant de cerner clairement les problèmes. En outre, ce système permet de traiter des cas qui, jusqu'à présent, étaient restés sans réponse. Ce travail est décrit dans [61].

3.3.4 Interprétation des preuves comme programmes

Participant : Christine Paulin

Les preuves du Calcul des Constructions peuvent être interprétées comme des programmes certifiés à travers la définition d'une notion de réalisabilité. Celle-ci fournit une méthode d'extraction de programmes à partir de preuves ainsi qu'une interprétation sémantique des formules comme ensemble de programmes qui permet de justifier des extensions du système. L'interprétation utilisée par Coq avait été définie pour un calcul sans types inductifs ni univers et n'admet pas d'extension simple. Les nouvelles méthodes d'extraction initiées par Takayama et reposant sur une distinction entre variable de preuve calculatoire ou non s'étendent par contre au Calcul des Constructions Inductives. Nous avons défini une telle interprétation et étudié son intégration dans Coq. Celle-ci permet de justifier une extension conservative du formalisme dans laquelle il est possible de nommer et de raisonner sur le terme extrait d'une preuve, ce qui augmente le pouvoir expressif du langage et permet aussi de gagner en efficacité lorsque les preuves doivent être exécutées comme dans le cas des tactiques utilisant un principe de réflexion.

Une difficulté est d'adapter la méthode de synthèse de preuves à partir de programmes à cette interprétation. Une autre difficulté est de permettre la cohabitation dans le même système de preuves classiques et de preuves interprétées constructivement comme des programmes sans aboutir à un système incohérent.

3.4 Vers un système vérifié de traitement de démonstrations

Le système Coq n'est pas lui-même prouvé correct vis à vis de sa spécification. Du fait de ses caractéristiques: correction des preuves ramenée à un algorithme de typage, extraction de code ML à partir des preuves, c'est probablement l'assistant à la démonstration le mieux adapté à la concrétisation de ce défi.

Pour pouvoir spécifier et développer un système de traitement de démonstrations prouvé correct, il se pose le problème de concevoir un formalisme de preuves partielles comprenant des variables en attente d'être instanciées. La principale difficulté vient de l'interaction entre ces variables et les variables ordinaires du langage mathématique. Le formalisme des substitutions explicites permet de rendre compte de manière simple et élégante de cette interaction. Cela motive l'extension du Calcul des Constructions Inductives avec des substitutions explicites. C. Muñoz fait une présentation détaillée de ce sujet dans [40].

Un groupe de travail composé de B. Barras, C. Cornes, G. Dowek, H. Herbelin, G. Huet, H. Laulhère, C. Muñoz et B. Werner se réunit régulièrement pour discuter de l'extension du Calcul des Constructions Inductives avec des substitutions explicites, de la formalisation des propriétés de ce calcul dans le système Coq et de la possibilité de développer le noyau d'un système prouvé correct à partir de ces démonstrations.

3.4.1 Un vérificateur de types générique certifié

Participant : Bruno Barras

B. Barras a poursuivi son travail de formalisation de vérificateurs de preuves. Il a généralisé les preuves faites pour le *Calcul des Constructions* au cas plus vaste des *Systèmes de Types Purs (PTS)* avec sous-typage. Les résultats prouvés permettent de délimiter une classe de *PTS* pour lesquels la relation de typage est décidable grâce à un algorithme inspiré de celui du *Calcul des Constructions*. La modularité de ces preuves rend possible la conception du vérificateur de preuves avant que le système implémenté ne soit complètement figé.

Il a plus spécifiquement étudié le cas du *Calcul des Constructions Étendu*, muni de la cumulativité, qui peut-être vue comme une sorte de sous-typage. En admettant la normalisation forte de ce calcul, il a prouvé la décidabilité du typage. Ces preuves ont donné lieu à l'extraction du noyau d'un nouveau système de vérification de preuves. L'utilisabilité de ce programme a été nettement améliorée par la possibilité d'écrire des définitions. Cette extension a nécessité la formalisation de la δ-réduction.

3.4.2 Types dépendants et substitutions explicites

Participant : César Muñoz

C. Muñoz a continué son travail sur l'extension du Calcul de Types Dépendants avec substitutions explicites. Il a montré comment certains systèmes avec substitutions explicites n'admettent pas d'extensions compatibles avec cette théorie des types d'ordre supérieur.

Il propose un nouveau calcul de substitutions explicites dont l'extension aux types dépendants a les propriétés de typage usuelles: unicité, préservation sous la réduction et décidabilité. Ce travail doit encore être généralisé au Calcul des Constructions Inductives.

3.5 Étude et extensions du Calcul des Constructions

Nous poursuivont l'étude métathéorique du Calcul des Constructions inductives. En effet, la puissance du formalisme rend paradoxales certaines extensions qui peuvent par ailleurs sembler naturelles. J. Courant a développé un langage de modules au-dessus de ce calcul. E. Giménez s'est intéressé à l'étude de l'extension par des types co-inductifs, cela lui a suggéré une manière moins syntaxique de représenter les conditions de gardes des définitions par point fixe. G. Dowek étudie les liens entre théorie des types et théorie des ensembles pour proposer des formulations alternatives du Calcul des Constructions.

3.5.1 Calcul de module au-dessus des systèmes de types purs

Participant : Judicaël Courant

La lourdeur du formalisme est l'un des problèmes majeurs lors de la manipulation formelle de preuves. En particulier, il est très difficile de réaliser des bibliothèques de preuves qui soient à la fois utilisables et génériques. L'approche des systèmes de modules des langages de programmation à la SML semble de ce fait intéressante. Les avancées récentes en la matière, en particulier les travaux de Leroy, Harper et Lillibridge, permettaient d'envisager une adaptation de ces systèmes de modules aux langages de preuves. J. Courant s'est attaché à améliorer ces systèmes, de façon à ce qu'ils possèdent la propriété de préservation des types par le calcul (appelée propriété d'auto-réduction) sans perdre leur capacité d'abstraction. La propriété d'auto-réduction a permis d'étudier ces systèmes de modules sous l'angle des réductions. En particulier, J. Courant a montré que la réduction des modules était fortement normalisante. Le calcul ainsi obtenu s'adapte aux *Systèmes de Types Purs* (PTS). En particulier, il fournit une extension conservative du Calcul des Constructions, qui devrait permettre de mieux modulariser les développements Coq, ainsi que la vérification séparée des différents modules les composant.

3.5.2 Types coinductifs

Participant : Eduardo Giménez

Dans les programmes séquentiels, le but est de calculer une donnée de sortie à partir de certaines données d'entrée. Pour cela, la terminaison du calcul joue un rôle crucial. En revanche, pour des programmes dont le but est de contrôler ou maintenir une certaine interaction avec leur environnement (par exemple contrôler un certain processus physique externe à l'ordinateur) la terminaison n'est plus essentielle. Dans ce type de programmes (dits programmes réactifs), la considération de boucles d'interaction infinie nous amène très vite à considérer également des types de structures infinies (listes de données infinies, espaces d'états infinis, etc.). Cette constatation motive l'idée que la vérification de programmes réactifs nécessite un cadre logique dans lequel on est capable de décrire et de raisonner sur des objets infinis.

Une extension du *Calcul des Constructions Inductives* avec des types qui contiennent des éléments potentiellement infinis (dits types co-inductifs) avait déjà été implantée dans la version V5.10 du système par Eduardo Giménez. Cette implantation avait été testée par la vérification d'un interpréteur interactif de processus et un protocole de communication des données. Ces expériences ont permis de détecter certains point faibles dans l'implantation actuelle des types co-inductifs: par exemple, des mauvaises interactions entre la tactique pour construire des preuves infinies et les tactiques automatiques, ou l'impossibilité de combiner des définitions par récursion bien fondée et par co-récursion dans le même bloc de déclarations récursives.

L'utilisation d'une condition syntaxique de garde pour assurer la correction des définitions récursives semble être à la base d'une bonne partie de ces faiblesses. Sur le plan théorique, cette condition syntaxique, externe aux règles de typage, rend difficile l'étude des propriétés du calcul implanté. Eduardo Giménez a étudié une nouvelle extension du Calcul de Constructions avec des types inductifs et co-inductifs, où cette condition est mieux intégrée aux règles du calcul par des annotations dans le jugement de typage. Il a aussi étudié différentes propriétés métathéoriques de ce système, telles que la préservation du typage par réduction, la confluence, et la normalisation forte. Le calcul développé pourrait permettre une implantation plus souple et expressive des types (co)inductifs dans Coq.

3.5.3 Types inductifs

Participant : Christine Paulin

Dans son mémoire d'habilitation à diriger les recherches [32], C. Paulin a étudié la représentation des définitions inductives dans différentes logiques d'ordre supérieur. Elle a également formalisé les définitions utilisées dans Coq et montré des réductions entre différents schémas. Cette étude a pour

but la recherche d'un formalisme plus élémentaire que celui actuel des définitions inductives qui serait suffisant pour un codage efficace des définitions et serait donc un meilleur candidat pour une étude métathéorique implémentée.

3.5.4 Preuves de cohérence

Participant : Benjamin Werner

Prouver la cohérence logique d'un λ-calcul typé comme le formalisme de Coq passe en général par un résultat de normalisation forte. La complexité de Coq rendait, jusqu'à maintenant, ces preuves assez obscures. En collaboration avec Paul-André Melliès actuellement en séjour post-doctoral à l'université d'Edimbourg, Benjamin Werner a proposé une preuve de normalisation forte générique pour les systèmes de Types Purs [62] qui clarifie le raisonnement en en séparant nettement les différentes étapes.

Les techniques utilisées dans ce travail s'étendent agréablement au types inductifs, ce qui doit permettre une première preuve de cohérence complète du formalisme de Coq.

3.5.5 Une nouvelle formulation de la théorie des types

Participant : Gilles Dowek

G. Dowek a poursuivi son travail sur la reformulation du Calcul des Constructions dans un cadre non typé. Il a proposé [49] une théorie non typée du premier ordre qui est une formalisation des mathématiques dans laquelle les démonstrations sont des objets. La cohérence de cette théorie est ouverte. Ce travail a été présenté au "Logic Colloquium" et au symposium "Types".

3.6 Structures fondamentales de calcul

La représentation interne des objets manipulés joue un rôle essentiel dans l'architecture d'un assistant à la démonstration permettant une meilleure compréhension logique des structures opérationnelles.

G. Huet a investigué un certain nombre de structures de données fondamentales pouvant servir de représentation à des structures de preuve. G. Dowek a poursuivi l'analyse des algorithmes d'unification d'ordre supérieur à la lumière des calculs avec substitution explicite. Enfin Ph. Audebaud et H. Herbelin s'intéressent aux liens entre les formalismes calculatoires associés à logique classique et les traits impératifs des langages de programmation fonctionnels.

3.6.1 Zippers

Participant : Gérard Huet

Les "zippers" sont des structures de données applicatives permettant la représentation de structures mutables, avec accès et mutation en temps constant. Il y a une manière uniforme de transformer la signature d'une algèbre libre en l'algèbre de "zippers" correspondante. Ce travail peut servir de base à la conception d'un éditeur structuré de structures arborescentes, l'espace de navigation et d'édition étant le zipper approprié.

Ce travail a donné lieu à présentation à la conférence invitée plénière commune aux conférences LICS et RTA, dans le cadre du congrès fédérateur FLoC'96, à New-Brunswick (NJ) en juillet 1996 et fait l'objet d'un article à paraître [35].

3.6.2 Arbres de Böhm rationnels

Participants : Gérard Huet, Henri Laulhère

Les arbres de Böhm, formes normales à l'infini des λ-termes, peuvent être présentés par des systèmes récursifs de combinateurs. Un tel système fini présente un arbre de Böhm rationnel, au sens d'avoir

un nombre fini de sous-arbres distincts. L'égalité extensionnelle de ces arbres (ou égalité dans le modèle D_∞) est décidable par un algorithme de mémorisation d'équivalences rationnelles. Ce travail a fait l'objet d'une conférence invitée au colloque "Mathematical Foundations of Programming Semantics" (MFPS'96) à Boulder (Colorado) en juin 1996, d'une conférence invitée au colloque "Logique et modèles du calcul" au CIRM (Luminy) en septembre, et d'une conférence au séminaire de logique de Paris-7 en octobre.

Les arbres de Böhm rationnels permettent de représenter des algorithmes réactifs sur des structures de données co-inductives. En particulier, on peut y coder les primitives du langage de programmation réactif Lustre. Henri Laulhère investigue cette traduction, ainsi que les problèmes algorithmiques liés à l'implémentation efficace de l'algorithme de décision, dans le cadre de sa thèse.

3.6.3 Unification d'ordre supérieur

G. Dowek, Th. Hardin (LITP et Projet Para) et C. Kirchner (du projet Prothéo) ont poursuivi leur travail sur l'utilisation du calcul des substitutions explicites pour simplifier la formulation des algorithmes d'unification dans le λ-calcul simplement typé. En collaboration avec F. Pfenning de l'Université Carnegie Mellon (États-Unis), ils ont donné un algorithme pour un sous-cas décidable de l'unification appelé "unification des motifs". Cet algorithme a été implémenté dans l'interprète du langage Elf développé par F. Pfenning. Ce travail a été publié dans les actes de la conférence "Joint International Conference and Symposium on Logic Programming" [38].

3.6.4 $\lambda\mu$-calcul et compilation des langages fonctionnels

Participant : Philippe Audebaud

Le $\lambda\mu$-calcul de M. Parigot de l'université de Paris 7 est un bon candidat pour étendre l'isomorphisme de Curry-Howard à la logique classique. L'étude menée sur ses aspects calculatoires a permis de montrer, via une extension aisée du calcul, qu'il intègre également les continuations comme objets de première classe.

A ce stade, Ph. Audebaud a obtenu un calcul partageant les mêmes "bonnes" propriétés du λ-calcul pur : confluence, finitude des développements, etc. De plus, une partie stable de ce calcul correspond au code naturellement compilé vers la machine de Krivine (par exemple). Il est ainsi possible d'exprimer et de vérifier dans le même temps certaines simplifications de programmes réalisées lors de la compilation, ainsi que la correction de la machine abstraite qui exécute le code compilé.

Une seconde extension du calcul, inspirée de travaux de Danvy, Filinski et Moggi, a débouché sur un calcul offrant la possibilité de prendre en compte diverses stratégies d'évaluation rencontrées dans les langages fonctionnels. De tout cela, Ph. Audebaud a pu tirer un modèle mathématique adapté pour la compilation et l'évaluation de programmes composant le noyau fonctionnel de Caml.

A ce point de l'étude, il est déjà possible d'écrire des coroutines par exemple, car les affectations ne sont pas nécessaires. Cependant, le traitement des exceptions, pas plus que celui des références n'est possible. Une dernière extension a donc été étudiée, qui a débouché sur un modèle, c'est-à-dire un calcul, permettant la représentation de programmes Caml impératifs et/ou utilisant des exceptions. Cette partie est basée sur des idées originales de Felleisen relatives aux références et aux prompts.

Un prototype a été réalisé qui met toutes ces idées en pratique. L'ensemble de l'étude a porté à la fois sur les termes concrets et sur une représentation avec substitutions explicites basée sur le lambda-sigma-lift calcul de Abadi, Cardelli, Curien, Lévy et Hardin. L'étude du typage, exclue de l'approche précédente, est maintenant abordée.

3.6.5 Modèles du λ-calcul

Participant : Hugo Herbelin

Hugo Herbelin a étudié les formalismes en terme de jeux du calcul, en particulier dans leurs rapports avec les machines à environnements [37, 58, 57].

Cela a conduit à de nouvelles machines à environnements pour la réduction d'expressions fonctionnelles en manipulant des arbres de Böhm plutôt que des lambda-termes [55].

3.7 Verified Internet Protocols

Participants : Jean-Christophe Filliâtre, Jean Goubault-Larrecq, Gérard Huet

G. Huet est le responsable INRIA du projet VIP (Verified Internet Protocols), où participent, avec le projet Coq, les projets Cristal de Rocquencourt, Lande de Rennes et Eureca de Nancy, ainsi que l'équipe de Dominique Bolignano du GIE Dyade. Dans ce cadre, il a conduit une étude sur les modèles de sécurité de code mobile. Cette action a mené à la conception d'un modèle de code mobile auto-certifiant mettant en œuvre la technologie d'extraction de programmes de preuves Coq. Cette première étude mène à une réflexion en commun avec le projet Cristal sur la définition d'une couche basse de représentation des programmes Objective Caml qui servirait à décrire la sémantique de langages de programmation impératifs.

Cette étude n'est que l'avant-projet d'une nouvelle direction de recherche commune aux projets Coq et Cristal qui semble prometteuse pour les applications liées à la certification de protocoles de paiement basés sur la technologie Internet, et plus généralement au Commerce Electronique. Nous estimons notre collaboration avec Dyade tout à fait stratégique du point de vue de nos objectifs de valorisation du produit Coq.

Dans le cadre du projet VIP, J. Goubault-Larrecq a conçu un langage formel minimaliste de description de protocoles cryptographiques, *slap*. Ceci a divers buts, celui pertinent au projet Coq est la possibilité de produire des morceaux de code réalisant les différentes parties du protocole qui pourront être certifiés.

4 Actions industrielles

4.1 Dassault : Projet Génie

Nous avons une collaboration suivie avec E. Ledinot de Dassault Aviation, au sein de la convention *Génie* entre Dassault et l'INRIA. Cette collaboration concerne en particulier les aspects de développement de programmes impératifs, de modélisation de systèmes communicants à l'aide de définitions inductives et la construction d'outils de preuve de haut-niveau.

4.2 GIE Dyade

Le projet de code mobile auto-certifié dans le cadre de la vérification de protocoles est en coopération avec l'équipe de D. Bolignano des laboratoires de recherche de Bull au sein de l'action VIP du GIE Dyade (Bull-INRIA). D. Bolignano utilise Coq pour la formalisation de protocoles sécuritaires pour le Commerce Electronique.

4.3 CNET

Nous avons aussi une collaboration suivie avec P. Crégut et J.-F. Monin du CNET à Lannion dans le cadre d'un contrat de trois ans auquel participe également le projet Croap. Ce contrat porte sur le développement dans Coq d'outils de réutilisation de preuves et d'une interface permettant l'écriture de

tactiques de haut niveau. Le CNET s'intéresse à Coq pour la validation de protocoles de communication. P. Crégut en collaboration avec B. Heyd du projet Eureca à Nancy a développé un fragment significatif du langage Unity dans Coq permettant de s'attaquer à la preuve de protocoles.

5 Actions nationales et internationales

5.1 Actions nationales

Nous formons un groupe du GDR/PRC Programmation en collaboration avec l'équipe de J.-F. Monin au CNET et l'équipe de Thérèse Hardin et Véronique Donzeau-Gouge à Paris 6 et au CNAM.

Nous participons également au GDR/PRC Algorithmique, Modèles et Infographie.

5.2 Actions internationales

5.2.1 Europe de l'ouest

Le *Working Group* ESPRIT "TYPES" a démarré officiellement le 15 septembre. Il porte sur le développement assisté par ordinateur de preuves et de programmes et regroupe principalement des équipes de Cambridge, Edinburgh, Göteborg, Helsinki, Manchester, Munich, Nijmegen, Oxford, Paris 7, Turin, Udine et Utrecht. Ce groupe est piloté par un comité de trois personnes. Gérard Huet faisait partie de ce comité la première année et sera remplacé par Christine Paulin.

6 Diffusion des résultats

6.1 Actions d'enseignement

Les activités d'enseignement de P. Audebaud, J. Courant et J. Duprat se font dans le cadre de leur service à l'Ecole Normale Supérieure de Lyon.

B. Barras est chargé de Travaux Pratiques dans le cours d'Algorithmique de 2ème année de DEUG à l'université Paris 7.

Samuel Boutin était chargé d'exercices dirigés pour le cours de programmation "CAML et ADA" au Conservatoire National des Arts et Métiers.

G. Dowek a donné un cours "Preuves constructives" dans le DEA "Sémantique, Preuves et Programmation", un cours "Démonstration automatique" à l'Ecole Nationale Supérieure des Techniques Avancées qui a donné lieu à la rédaction de notes. Il a participé au jury des stages de fins d'études à l'Ecole Nationale Supérieure des Techniques Avancées.

G. Dowek et C. Paulin ont donné un cours à l'école des jeunes chercheurs du GDR Programmation à Bordeaux.

J. Duprat est examinateur a l'épreuve d'informatique du concours d'entrée de l'Ecole Polytechnique.

H. Herbelin a été chargé d'enseignement à l'université Paris 7 jusqu'en septembre 1996. A partir d'octobre 1996, il a enseigné à l'université de Nanterre. Il a dispensé en licence d'informatique des travaux dirigés de logique propositionnelle et de programmation fonctionnelle basée sur Caml-Light. Il a dispensé des travaux dirigés de logique en Deug de psychologie et de programmation Pascal en Deug MASS.

H. Laulhère était chargé de TD pour le cours d'algorithmique de première année de l'Ecole Polytechnique.

C. Paulin a donné un cours "Types et Preuves" au DEA d'Informatique de Lyon et un cours sur les langages fonctionnels typés en magistère d'informatique à l'ENS Lyon.

Amokrane Saïbi était chargé de TD pour le cours de programmation "CAML et ADA" au Conservatoire National des Arts et Métiers. Pendant le premier semestre de 96-97, A. Saïbi était chargé de TD et TP du module d'Algorithmique en licence d'informatique, et de programmation Pascal en DEUG.

B. Werner enseigne l'utilisation de Coq en enseignement thématique d'informatique à l'Ecole Nationale Supérieure des Techniques Avancées.

6.2 Encadrement

P. Audebaud encadre les travaux de thèse de J. Courant et F. Prost.

G. Dowek encadre les travaux de thèse de C. Muñoz.

H. Herbelin a partiellement encadré à l'université Paris 7 le travail de thèse de T. Crolard.

G. Huet encadre les travaux de thèse de S. Boutin, C. Cornes, H. Laulhère et A. Saïbi.

C. Paulin encadre les travaux de thèse de E. Giménez et J.-C. Filliâtre.

B. Werner encadre les travaux de thèse de B. Barras.

6.3 Jurys

C. Paulin a été rapporteur de la thèse de O. Ait-Mohamed (Nancy). Elle a participé comme expert à un jury de recrutement post-doctoral à Göteborg.

C. Paulin a fait partie du jury de recrutement CR à l'INRIA Rocquencourt et Gérard Huet du jury de recrutement CR à l'INRIA Rhones-Alpes.

6.4 Responsabilités éditoriales et professionnelles

C. Paulin est responsable du pôle "Preuves et Spécifications Algébriques" du GDR Programmation.

G. Huet est membre de "Academia Europæ" et membre correspondant de l'Académie des Sciences. Il 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". Il est membre du Comité de Pilotage du GIE-Dyade, et du Conseil Scientifique de VERIMAG. Il est membre du groupe de travail "Systèmes Critiques" à l'OFTA. Il est membre du "Honorary Board" du RIDS (Research Institute for Declarative Systems) à Nijmegen. De janvier à juillet 1996, G. Huet a été Conseiller auprès de la Direction du développement. À partir de Janvier 1997 il sera Délégué aux Affaires Internationales à l'INRIA.

G. Huet a participé au Conseil Scientifique de VERIMAG à Grenoble.

G. Dowek est membre du comité de programme du symposium "Higher Order Algebra 97" et de la conférence "Typed Lambda-Calculi and Applications 97".

C. Paulin est membre du comité de programme de TPHOL'96 et TPHOL'97. Elle a été rédacteur invité d'un numéro spécial TSI. Elle a organisé le Comité Scientifique du PRC/GDR Programmation à Lyon. Elle est membre du groupe de réflexion prospective "Maîtrise des systèmes complexes réactifs et sûrs" commun au CNRS à la DRET et au MENESR.

Ph. Audebaud et C. Paulin ont organisé en collaboration avec René David de l'université de Chambéry une journée "Logique Classique pour le Programmeur" dans le cadre du PRC/GDR AMI.

E. Giménez et C. Paulin organisent la rencontre internationale annuelle TYPES'96.

6.5 Conférences et visites

Les membres de l'équipe ont largement participé aux conférences du domaine.

6.5.1 Conférences internationales

Type Theory and Term Rewriting Ph. Audebaud, C. Muñoz, C. Paulin et F. Prost ont participé à l'école "Type Theory and Term Rewriting" à Glasgow (Écosse). C. Paulin y a donné un exposé invité.

TYPES Ph. Audebaud, S. Boutin, C. Cornes, G. Dowek, J. Duprat, J.-C. Filliâtre, E. Giménez, H. Herbelin, C. Muñoz, Ch. Paulin et B. Werner ont participé au symposium annuel Types'96 à Aussois (France). B. Barras, S. Boutin, C. Cornes, G. Dowek, H. Herbelin et B. Werner y ont présenté des exposés.

TPHOL S. Boutin, C. Paulin, A. Saïbi ont participé à la conférence TPHOL'96 qui s'est tenue à Turku en Finlande du 24 au 30 août. C. Paulin y a présenté un tutorial.

FST-TCS G. Huet a participé à un atelier "Specification and Certification of Critical Systems" à la conférence FST/TCS à Hyderabad.

FLOC G. Huet a été conférencier invité au congrès fédérateur FLoC'96, à New-Brunswick (NJ) en juillet.

C. Muñoz a participé aux conférences "Logic in Computer Science" et "Automated Deduction", à New Brunswick (États unis). Il a présenté un exposé à la conférence "Logic in Computer Science".

MFPS G. Huet a été conférencier invité au colloque "Mathematical Foundations of Programming Semantics" (MFPS'96) à Boulder (Colorado).

ICSLP G. Dowek a présenté un exposé à la conférence "Joint International Conference and Symposium on Logic Programming" à Bonn (Allemagne)

Logic Colloquium G. Dowek a présenté un exposé au "Logic Colloquium" à San Sebastian (Espagne).

UNIF G. Dowek a participé au symposium "Unif" à Munich (Allemagne)

6.5.2 Animation nationale

Logique Classique pour le programmeur Ph Audebaud, J. Courant, J. Duprat, E. Giménez, et C. Paulin ont participé à la journée "Logique Classique pour le Programmeur" organisée à Lyon. Ph Audebaud a présenté un exposé intitulé : "Une étude autour des exceptions de Caml-light" et H. Herbelin a présenté un exposé intitulé : "Jeux et logique classique".

Journées des GDR B. Barras, S. Boutin, J. Courant, C. Cornes, E. Giménez, J.-C. Filliâtre, H. Laulhère, C. Muñoz, C. Paulin, et B. Werner ont participé aux journées du GDR de Programmation à Orléans en novembre. S. Boutin, J. Courant, H. Laulhère, C. Muñoz y ont présenté un exposé et C. Cornes une démonstration.

G. Dowek a participé aux journées "Lambda-calcul et réecriture" du GDR AMI à Orsay, où il a présenté un exposé.

J. Duprat a participé aux journées du PRC Architectures Nouvelles de Machines à Rennes en février.

Systèmes Critiques C. Paulin a été conférencier en janvier au Colloque sur l'application des méthodes formelles au développement des systèmes critiques à Grenoble.

6.5.3 Divers

Ecole d'été Marktoberdorf B. Barras a participé à l'école d'été Marktoberdorf School on Mathematical Methods in Program Development.

W3C G. Huet a participé à la conférence W3C update au CNIT.

Network Security G. Huet a participé à la conférence "Network Security" à New-Delhi.

Colloque Renou G. Huet a participé au colloque Renou à l'ENS en janvier.

Panel CLEI 96 C. Muñoz a participé à la conférence latino-americaine d'informatique "Panel CLEI 96" à Santafé de Bogotá (Colombie) où il a présenté un exposé.

Verification day C. Paulin a donné une conférence lors d'une journée "Verification day" organisée par l'université technologique de Eindhoven.

Logique et Modèles du Calcul G. Huet a été au colloque "Logique et modèles du calcul" au CIRM (Luminy).

LFCS G. Huet a été conférencier invité au colloque du 10ème anniversaire du Laboratory on Foundations of Computer Science (LFCS) à l'Université d'Edimbourg.

Academia Europæ G. Huet a participé à un symposium du groupe des informaticiens d'Academia Europæ à Amsterdam.

Table Ronde H. Herbelin a participé à la table-ronde sur les jeux à Imperial College (Londres) en novembre.

Visites Industrielles G. Huet a participé à la présentation Génie chez Dassault, et à des rencontres INRIA-Aérospatiale, à Toulouse et aux Mureaux . Il a rendu visite à la société Bertin en avril. Il a participé à la réunion annuelle VIP à Nancy en septembre.

Visites de courte durée C. Muñoz a passé une semaine à l'université EAFIT de Medellín (Colombie) où il a donné un cours.

B. Werner a visité le laboratoire d'informatique de l'Université d'Edimbourg où il a donné un exposé de séminaire et le Laboratoire de Mathématiques Discrètes de Marseille-Luminy.

6.6 Diffusion de logiciels

L'équipe a mis dans le domaine public la version 6.1 de l'assistant à la démonstration Coq.

7 Publications

Thèses

[31] E. GIMÉNEZ, *Un Calcul de Constructions Infinies et son application a la vérification de systèmes communicants*, thèse de doctorat, Ecole Normale Supérieure de Lyon, décembre 1996.

[32] C. PAULIN-MOHRING, *Définitions Inductives en Théorie des Types d'Ordre Supérieur*, Habilitation à diriger les recherches, Université Claude Bernard Lyon I, décembre 1996.

[33] A. SAÏBI, *Algèbre Constructive en Théorie des Types, Outils génériques pour la modélisation et la démonstration, Application à la théorie des Catégories*, thèse de doctorat, Université Paris VI, 1997.

Articles et chapitres de livre

[34] G. HUET, A. SAÏBI, *Constructive Category Theory*, à paraitre 1996.

[35] G. HUET, «The Zipper», *Journal of Functional Programming*, 1997, à paraitre.

Communications à des congrès, colloques, etc.

[36] C.CORNES, D.TERRASSE, «Inverting Inductive Predicates in Coq», *in : BRA Workshop on Types for Proofs and Programs (TYPES'95)*, *LNCS*, *1158*, 1996.

[37] V. DANOS, H. HERBELIN, L. REGNIER, «Game Semantics and Abstract Machines», *in : Proceedings, Eleven Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society Press, New Brunswick, New Jersey, July 1996.

[38] G. DOWEK, T. HARDIN, C. KIRCHNER, P. PFENNING, «Higher-order unification via explicit substitutions : the case of patterns», *in : Joint International Conference and Symposium on Logic Programming*, 1996.

[39] E. GIMÉNEZ, «An application of co-Inductive types in Coq: verification of the Alternating Bit Protocol», *in : BRA Workshop on Types for Proofs and Programs (TYPES'95)*, *LNCS*, *1158*, 1996.

[40] C. MUÑOZ, «Confluence and Preservation of Strong Normalisation in an Explicit Substitutions Calculus (Extended Abstract)», *in : Proceedings, Eleven Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society Press, New Brunswick, New Jersey, juillet 1996.

[41] C. MUÑOZ, «Proof Representation in Type Theory: State of the Art», *in : Proceedings, XXII Latinamerican Conference of Informatics CLEI Panel 96*, Santafé de Bogotá, Colombia, juin 1996.

[42] C. PAULIN-MOHRING, «Circuits as streams in Coq : Verification of a sequential multiplier», *in : Types for Proofs and Programs (TYPES'95)*, S. Berardi, M. Coppo (réd.), *LNCS*, *1158*, 1996. également Rapport de Recherche LIP 95-16.

[43] A. SAÏBI, «Typing algorithm in type theory with inheritance», *in : Proceedings of The 24th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ACM Press, Paris, France, janvier 1997.

Cours polycopiés

[44] G. DOWEK, *Démonstration automatique*, Ecole Nationale Supérieure des Techniques Avancées, 1996.

Rapports de recherche et publications internes

[45] P. AUDEBAUD, C. PAULIN-MOHRING, F. PROST, «Informative contents as annotations in the Calculus of Inductive Constructions», *rapport de recherche*, LIP ENS-Lyon, 1996.

[46] B. BARRAS, «Coq en Coq», *rapport de recherche*, INRIA, Octobre 1996.

[47] C. CORNES, J. COURANT, J.-C. FILLIÂTRE, H. H. G. HUET, P. MANOURY, C. MUÑOZ, C. MURTHY, C. PARENT, C. PAULIN-MOHRING, A. SAÏBI, B. WERNER, «The Coq Proof Assistant Reference Manual, Version 6.1», *rapport technique*, INRIA-ENS Lyon, décembre 1996, Disponible en ftp avec la distribution Coq.

[48] J. COURANT, «A module calculus enjoying the subject-reduction property», *rapport de recherche*, LIP, 1996, Preliminary version.

[49] G. DOWEK, «A type-Free formalization of mathematics where proofs are objects», *Rapport de Recherche n°2915*, INRIA, 1996.

[50] J.-C. FILLIÂTRE, «A decision procedure for Direct Predicate Calculus: study and implementation in the system COQ», *rapport de recherche n°96–25*, LIP - ENS Lyon, 1996.

[51] G. HUET, G. KAHN, C. PAULIN-MOHRING, «The Coq Proof Assistant - A tutorial», *rapport technique n°178*, INRIA, Juillet 1995, Version révisée distribuée avec Coq.

[52] C. MUÑOZ, «Confluence and Preservation of Strong Normalisation in an Explicit Substitutions Calculus», *rapport de recherche n°RR-2762*, Unité de recherche INRIA-Rocquencourt, décembre 1995.

[53] F. PROST, «Marking Techniques for Extraction», *rapport de recherche n°95-47*, Laboratoire de l'Informatique du Parallélisme, ENS Lyon, décembre 1995.

[54] A. SAÏBI, «Une axiomatisation constructive de la théorie des catégories», *rapport de recherche*, INRIA, 1995, en préparation.

Divers

[55] P.-L. CURIEN, H. HERBELIN, «Computing with Abstract Böhm Trees», soumis, 1997.

[56] E. GIMÉNEZ, «A tutorial on recursive types in Coq», Distribué comme partie de la documentation du système Coq., Mars 1996, Notes de cours rédigées pour la formation Coq qui s'est tenu à l'ENS-Lyon en Mars 1996.

[57] H. HERBELIN, «From HO-games to AJM-games», Présenté à la table-ronde sur les jeux, Novembre 1996.

[58] H. HERBELIN, «Games and Weak Head Reduction for Lambda-Calculus + Catch», soumis, 1997.

[59] G. HUET, «Regular Böhm Trees», En préparation, 1996.

[60] F. PROST, P. AUDEBAUD, «A Unifying framework for extraction», soumis, 1996.

[61] F. PROST, «Dead code Elimination, Typing and Overloaded Functions», soumis, août 1996.

[62] B. WERNER, P. MELLIÈS, «A generic normalization proof for type systems», soumis, 1996.

8 Abstract

The goal of the Coq project is to design proof processing systems, i.e. systems verifying, producing and transforming mathematical proofs. We are developing such a system called Coq, we are using this system to develop proofs (in particular correction proofs of software designs), we are studying the formalized mathematical language implemented in this system and some alternatives.