Activités d'enseignement et de recherche d' Olivier PONS

Projet Croap

INRIA Sophia Antipolis
2004 route des lucioles
BP 93
06902 Sophia Antipolis
Olivier.Pons@sophia.inria.fr

1 Présentation générale

Je suis inscrit en thèse au *Conservatoire National des Arts et Métiers*, spécialité *Informatique*, depuis la rentrée universitaire 1995-96. Cette thèse, dont le sujet est "**Conception et réalisation d'outils d'aide au développement de grosses théories dans les systèmes de preuves interactifs**" est financée depuis le 15 Novembre 1995 par une bourse INRIA sur un contrat CNET. Elle se déroule dans le cadre du projet Croap de l'INRIA Sophia Antipolis sous la codirection de Laurence Rideau et d'Yves Bertot pour l'INRIA et de Véronique Donzeau-Gouge pour le CNAM.

L'objectif du projet Croap est d'étudier les techniques et les outils nécessaires pour produire du logiciel complexe et fiable, et la collaboration de ces outils dans des environnements de travail conviviaux et efficaces. Les thèmes principaux sont l'étude formelle des langages de programmation, la réalisation d'outils adaptés à des classes de langages spécifiques et l'utilisation des outils de preuve sur ordinateur pour assurer la validité des outils et des programmes produits.

Ma thèse s'inscrit principalement dans la partie touchant les environnements de travail autour des systèmes de preuve.

Il s'agit de fournir des outils permettant de mieux comprendre de grosses preuves, de gérer les dépendances entres des sous-preuves ou entre les différents résultats mathématiques afin d'en faciliter le développement et la maintenance. En effet, l'utilisateur d'un système de preuve effectue son travail par tâtonnements à partir de données initiales et d'un plan de travail imprécis. L'ensemble des outils développés permet une amélioration de la compréhension, de la maintenance, et la réutilisation possible des ébauches successives.

2 Stage de DEA

Mon stage de DEA, a été réalisé en 1995, au sein du Laboratoire d'Informatique Théorique de l'université Pierre et Marie Curie (Paris 6) sous la direction de Thérèse Hardin et Pascal Manoury. Son objectif était l' **Etude de spécifications des bases de données relationnelles en Coq**. Cela m'a permis d'acquérir un bonne connaissance de l'assistant de démonstration *Coq* et des problèmes et difficultés rencontrés lors des activités de développement dans ce système.

3 Travail de Thèse

Mon travail de thèse a pleinement profité de l'expérience personnelle acquise en DEA, ainsi que des expériences du projet Croap, dans le domaine des interfaces de preuve et de programmation comme dans celui de la vérification mécanique de preuves complexes. La preuve de l'algorithme de Buchberger, développé par Laurent Thery, représente plus de 9000 lignes de script Coq, 120 définitions et 400 théorèmes et le développement s'est étalé sur un an. Elle a mis en evidence le fait que l'organisation de la librairie de théorèmes revêt une importance majeure. Ainsi, l'ensemble des théorèmes relevant de l'arithmétique de base des polynômes doit être réuni au même endroit, les propriétés topologiques à un autre endroit, etc. L'organisation des théories peut influer de façon importante sur la vitesse de développement des preuves et leur réutilisabilité, deux facteurs importants pour que le système de preuve soit praticable en milieu industriel. Ces impératifs d'organisation imposent que le développement d'une théorie n'est pas une activité parfaitement linéaire.

Le développement d'une preuve, surtout sur machine, comporte de nombreux essais, erreurs et retours en arrière. Il importe de fournir les outils qui permettent de faciliter ces retours en arrière, pour éviter que ceux-ci ne deviennent si coûteux que l'utilisateur préférerait construire une théorie bancale plutôt que de faire l'effort de la nettoyer. Les preuves de compilateur réalisées par Yves Bertot dans CtCoq, ont pleinement montré la nécessité de tels outils. Il a d'ailleurs fait un usage intensif de ceux que nous proposons.

La première partie de mon travail s'est située à un niveau local, c'est-à-dire centré sur une preuve donnée. Dans un second temps, j'ai étudié le niveau global, c'est-à-dire les relations entre les différents théorèmes, définitions et théories. La notion fondamentale qui est à la base de ce travail est celle de dépendances et ses applications aux tâches de développement et de maintenance.

La majeure partie des expériences s'est faite pour le système Coq mais nous avons toujours tenté de définir des méthodologies applicables à d'autres systèmes d'aide à la preuve (comme HOL, Lego ou PVS).

Dans notre travail, nous avons tenté d'allier les avantages de la théorie des types, sur laquelle est basé Coq, (en particulier l'existence d'objets preuve) et des procédures de preuve. Si de nombreuses analyses pertinentes peuvent être faites sur les objets preuve, l'apport des tactiques est également indéniable en ce qu'il permet d'intégrer des composantes de preuve automatique. Ce point de vue hybride fournit des contraintes nouvelles pour certains problèmes (comme le problème du retour arrière logique).

Au niveau local:

Navigation dans les preuves. La preuve dirigée par les buts est une méthode de preuve par laquelle l'utilisateur énonce une formule à prouver et applique des commandes sur cet énoncé pour fournir de nouveaux énoncés, les sous-buts, et répétitivement jusqu'à ce que les sous-buts soient triviaux. De façon assez naturelle, un sous-but dépend d'un sous-but précédent par l'intermédiaire de la commande qui a été appliquée sur ce dernier. Ces dépendances fournissent une structure d'arbre. Par symétrie, une commande de preuve peut dépendre d'une précédente par l'intermédiaire du sous-but produit par cette dernière auquel cette commande s'applique.

La principale trace qu'un utilisateur conserve habituellement d'un développement de preuve est la séquence de commandes dans l'ordre où elles ont été envoyées au système de preuve. Dans cette séquence, les informations de dépendance décrites plus haut sont noyées dans la notion plus forte de dépendance chronologique. Nous avons étudié les moyens de redonner à l'utilisateur l'accès à ces dépendances logiques.

La méthode que nous avons mise au point passe par la maîtrise de la communication entre le système de preuve et l'éditeur où sont préparées et conservées ces commandes. Cette maîtrise permet d'abord d'assurer que les commandes sont bien rangées dans l'ordre où elles ont été envoyées. Ensuite, elle permet d'annoter (de façon invisible) chaque commande avec une information indiquant la place de cette commande dans l'arbre des dépendances.

Retour en arrière logique. La recherche d'une preuve n'étant pas une activité linéaire mais s'effectuant par tâtonnements et retours en arrière fréquents, il importe d'avoir un outil de retour en arrière efficace. Dans sa version actuelle, le système Coq comme la majorité des prouveurs fournit la possibilité de revenir en arrière en utilisant l'ordre chronologique. Ainsi, il est possible de défaire la dernière commande effectuée, ou bien les n dernières, pour un nombre relativement petit (n = 12, par défaut).

Dans la mesure où deux commandes successives peuvent s'appliquer sur des buts indépendants, on pourrait envisager de défaire l'avant-dernière commande sans défaire la dernière. Nous avons implementé un *undo logique* qui permet d'effacer une sous-preuve sans modifier les sous-preuves logiquement indépendantes. Cela se fait en coupant la branche de l'arbre de preuve correspondant. Le script contenu dans l'éditeur est alors réarrangé de façon a garder sa cohérence. En présence de variables existentielles, les branches de l'arbre de preuve ne sont plus forcement indépendantes les unes des autres et l'on peut être conduit à modifier plus de branches que celle que l'on veut couper. Pour savoir quelles sont les autres branches qui doivent être effacées, on est amené à considérer en plus des dépendances logiques issue de la structure, les dépendances historiques entre les sous-buts contenant les mêmes variables existentielles.

Travail sur des fragments de preuve et de lemme. Il s'agit d'isoler une sous-preuve pour en faire un lemme à part entière afin de le réutiliser ailleurs. Le principal problème consistant à déterminer et a gérer les hypothèses nécessaires à la sous-preuve.

De même, il arrive que dans une preuve on ait à prouver plusieurs fois les mêmes résultats (ou des résultats semblables), on veut alors être capable de détecter ces duplications pour les factoriser au moyen d'un `Cut` ou en introduisant un lemme intermédiaire.

Les outils de navigation et de retour en arrière sont maintenant fournis dans la distribution de CtCoq et ont fait l'objet d'une présentation au colloque UITP (User Interfaces for Theorem Provers) en septembre 97 à Sophia Antipolis. La fonctionnalité permettant d'isoler une sous preuve, fonctionne mais n'a pas encore été intégré dans Ctcoq. La factorisation de preuve est à l'état de prototype.

Calcul de dépendance au niveau global et maintenance de théories.

Ce calcul permet de détecter les dépendances entre les différents théorèmes et définitions constituant une théorie puis de les utiliser ensuite pour fournir une aide à l'utilisateur dans ses activités de développement ou de maintenance.

Calcul des dépendances Une analyse syntaxique du script est insuffisante pour comprendre les relations entre les différents théorèmes d'une théorie. Pour comprendre ces relations, il faut étudier le terme de preuve accessible en Coq et sauvegardé dans les fichiers compilés.

Les variables libres présentes dans les objets preuves dénotent des objets dans l'environnement, elles peuvent représenter des objets précédemment définis ou des objets relatifs aux définitions inductives (le type lui-même ou l'un de ses constructeurs).

En analysant un terme donné on peut récupérer la liste des identificateurs dont il dépend. En appliquant cela récursivement on peut construire le graphe de dépendance entre les théorèmes d'une théorie.

Les dépendances fournies par les termes normalisés caractérisent les résultats qui sont indispensables pour pouvoir faire la preuve. Ils permettent d'affirmer qu'il existe une preuve n'utilisant que ces résultats. Mais ces dépendances sont insuffisantes pour affirmer que l'on pourra rejouer le script de preuve dans un contexte contenant exclusivement les identificateurs issus du calcul de dépendance. En effet, des identificateurs utilisés par les tactiques du script de preuve peuvent ne pas être réellement utiles à la preuve (par exemple si l'on a introduit un `Cut` inutile) et disparaître lors de la normalisation du terme de preuve. Les théorèmes associés à ces identificateurs ne sont pas utiles à la preuve mais doivent être présents dans le contexte pour exécuter le script.

Visualisation des dépendances. Il est appréciable de visualiser graphiquement les dépendances entre les objets d'une théorie. Pour cela j'ai interfacé notre calcul de dépendance avec les logiciels de dessin de graphes (Graphlet et daVinci). On utilise aussi cela pour visualiser les dépendances entre théories et les graphes d'héritage liés aux coercions implicites introduites dans Coq par A. Saibi (ces coercions permettent de simuler le sous-typage: Par exemple, une fois déclarée la fonction de coercion de groupe vers monoide on peut considérer un groupe comme un monoide sans avoir à se préoccuper des problèmes de typage.

Propagation de modifications. Il arrive que l'on doive modifier une axiomatique ou des résultats de base. Par exemple si notre objectif est de réutiliser une preuve dans un contexte plus général, on peut vouloir affaiblir les assertions sur lesquelles la preuve est basée. Il importe alors de savoir quelles parties du développement vont être touchées par ces modifications. Nous étudions des outils fournissant à l'utilisateur une aide interactive dans la propagation de ces modifications.

Déplacement de code et réorganisation de théorie. Enfi, il arrive que durant le développement d'une théorie, l'utilisateur ait besoin de prouver un résultat lié à une autre théorie. C'est-à-dire qu'en utilisant une théorie précédemment définie il découvre qu'il y manque un théorème important et qu'il le prouve dans la théorie qu'il est en train de développer. Pour éviter d'avoir trop de résultats "mal placés", il a donc parfois besoin de réordonner son développement et les théories. Nous étudions des outils aidant l'utilisateur dans cette tâche de déplacement de code.

L'ensemble de ces travaux autour de la notion de dépendances et de son utilisation dans les activités de développement et de maintenance a fait l'objet d'une présentation au colloque BRA Types 98 à Kloster Irsee en Allemagne et d'une soumission au colloque UITP 98.

Expériences autour de la généralisation de preuves.

Nous avons aussi mené quelques expériences autour de la notion de géneralisation de preuve. Il s'agit là encore d'utiliser les dépendances entre théorèmes pour comprendre quelles propriétés sont utilisées dans une preuve. L'idée est que lorsque l'on prouve une propriété d'un terme fonctionnel, on utilise un certain nombre d'autres propriétés de ce terme. On peut donc abstraire ce terme fonctionnel et ces propriétés et on obtient ainsi une preuve plus générale (valable pour tous les termes fonctionnels vérifiant ces propriétés).

Publication

Conférences internationales avec comité de lecture

[Pons 97] "Undoing and Managing a proof", actes de *User Interfaces for Theorem Provers 1997*, Septembre 1997, Sophia-Antipolis, France,
URL=http://www.inria.fr/croap/events/uitp97-papers.html

[Pons et al.98] "Notion of dependency in proof assistants", avec Y. Bertot et L. Rideau, soumis à *User Interfaces for Theorem Provers 1998*.

Autres Conférences

[Pons 96] "Outils de navigation et de maintenance dans un script de preuve", *Journée du GDR de programmation, Pôle Interfaces et Environnements*, Novembre 1996, Orléans, France

Applications développées

Outils de gestion de preuve Coq dans l'environnement Ctcoq
Outils de management de Script de preuve et de gestion de théories
Outils de visualisation des dépendances base sur le système de visualisation de graphe daVinci

Collaborations - Contacts

LIP6 (Paris VI) : Thérèse Hardin
CNAM (Paris) : Véronique Donzeau-Gouge
INRIA Rocquencourt et ENS Lyon : Equipes COQ
CNET Lanion :J.F. Monin

4 Enseignement

Vacations à l'Université de Nice Sophia Antipolis et à l'Ecole Supérieure en Sciences Informatiques (ESSI).

1996-97	TD-TP	*Sémantique des langages de programmation* (Cours d'Isabelle Attali) DEA d'Informatique de l'ESSI (12 h)
1996-97	TD	*Objets Formels pour l'Informatique* (cours de J.M. Fedou) Deug MP2 (24 h)
1996-97	TD-TP	*Scheme* (Cours de J.P Roy) Deug MI2 (26h)
1996-97	TD	*Java* Licence-Maîtrise MASS (26 h)
1997-98	Cours	*Java* Licence MASS (19h30
1997-98	TD	*Java* Licence MASS (43 h)
1997-98	TD-TP	*Scheme* (Cours de J.P Roy) Deug MI2 (26 h)

Le cours de sémantique était centré autour du système Centaur. Les TD et TP consistaient à construire un environnement de programmation pour *mini-ML*. Les étudiants commencaient par décrire la syntaxe en utilisant le langage *Metal*, puis l'affichage avec la description de pretty-printer en utilisant le langage *PPML*. Enfin la description sémantique en *sémantique naturelle* en utilisant le langage *Typol*.

Les TD et TD de *Scheme* consistaient en une introduction classique à la programmation avec *Scheme* comme support (à la maniere de ce qui est fait dans les livres d'Abelson et Susman ou de J. Chazarain.). On a par ailleurs proposé aux étudiants les plus motivés de réaliser de mini projets (jeux en utilisant l'algorithme min-max, mini système formel et introduction à la réécriture ...)

Le cours de *Java* s'adressait à des étudiants de Mathématiques et Sciences Sociales, aux connaissances informatiques diverses mais bien souvent minimales. On a néanmoins tenté de les initier aux problèmes du génie logiciel et à la programmation objet. On s'est librement inspiré de ce que B. Meyer avait fait pour le langage *Eiffel* dans son livre "Conception et programmation par objets ". On a veillé par ailleurs à maintenir la motivation en faisant de chaque TP un petit développement complet et autonome (illustrant chaque fois une notion nouvelle). Enfin on n'a pas négligé le côte "ludique" de Java en initiant les étudiants au développement d'applet et aux rudiments du graphisme. Cela nous a par ailleurs permis d'introduire en douceur la programmation par événement. Les TD sont visibles à : URL=http://www.mips.unice.fr\~opons

Enfin les TD d'OFI représentaient un survol rapide mais complet des bases de l'informatique théorique.