Bécassine à la chasse au Coq (démonstration)

Valentin Blot, Louise Dubois de Prisque, Chantal Keller, Pierre Vial

Bécassine à la chasse au Coq *

Démonstration d'un prototype

Valentin Blot[1,2], Louise Dubois de Prisque[1,2], Chantal Keller[1], and Pierre Vial[1,2]

[1] Université Paris-Saclay, CNRS, ÉNS Paris-Saclay, Laboratoire de Méthodes Formelles
[2] Inria

Résumé

Nous présentons une nouvelle tactique Coq (`snipe`) permettant de prouver *automatiquement* des buts Coq en les envoyant à des prouveurs automatiques externes du premier ordre *via* des plugins. Pour ce faire, la tactique `snipe` réduit le but à un énoncé du premier ordre et enrichit le contexte local avec des énoncés explicitant pour le prouveur la sémantique du but. Nous combinons des transformations modulaires et indépendantes permettant chacune de réduire un aspect spécifique du langage de Coq au langage (plus simple) d'un prouveur automatique. À l'heure actuelle, `snipe` utilise des transformations simples mais cruciales qui explicitent des définitions et des types algébriques. Ceci permet de prouver automatiquement des buts mêlant raisonnement au premier ordre, types de données et polymorphisme. Ce prototype de tactique automatique est un premier pas vers l'implantation et la combinaison de transformations plus complexes, qui rendront Coq plus facile d'accès.

1 Preuves automatiques et preuves interactives

Le manque d'automatisation de Coq—et des assistants de preuve en général—est un obstacle à la démocratisation de son utilisation auprès des mathématiciens ou dans l'industrie. La difficulté intrinsèque qu'il y a à automatiser Coq est étroitement liée à la richesse de son langage de spécification, basé sur le **Calcul des Constructions Inductives (CIC)**. À l'inverse, les prouveurs automatiques, qui manipulent des logiques moins expressives (en particulier, **logique du premier ordre**), ne reposent pas sur un *noyau* logiquement robuste (contrairement à Coq) mais ont des heuristiques de recherches de preuve très efficaces.

L'automatisation des démonstrations dans un cadre permettant les fonctions d'ordre supérieur, le polymorphisme et de types dépendants est un problème hautement non-trivial. En revanche, il est surprenant que **Coq n'arrive pas à automatiser la preuve de nombreux énoncés du premier ordre** sur des types décidables, alors qu'elle le serait sans problème dans des prouveurs du premier ordre. Même dans le cadre de la logique du premier ordre, les utilisateurs (particulièrement les débutants) doivent être très précis dans la façon dont ils écrivent leurs preuves : par exemple, il faut spécifier comment les variables des lemmes sont instanciées et régler de nombreux détails triviaux mais fastidieux, qui seraient ignorés sur papier.

Nous allons présenter :

(a) Une méthodologie générale, sur laquelle s'appuie notre prototype, pour faciliter l'appel à des prouveurs automatiques en Coq lorsqu'il s'agit de prouver des buts se réduisant à la logique du premier ordre. Comme nous le verrons, (1) nous prouvons automatiquement en Coq des énoncés de premier ordre variés et procédons à de légères transformations du but (2) nous envoyons tous les énoncés produits et le but à un prouveur externe

(b) Une implantation de cette méthodologie utilisant le plugin Coq SMTCoq à travers la tactique `snipe` (le nom albionique de la *Bécassine des Marais*).

À titre d'exemple, si nous voulons prouver l'énoncé du premier ordre suivant

```
Goal forall (A: Type) l (a:A), hd_error l = Some a → l <> nil.
```

une preuve Coq standard est `intros A l a H. intro H'. rewrite H' in H. simpl in H. inversion H.` Cette preuve repose sur le fait que les constructeurs d'un type inductif sont disjoints. Cependant, en Coq, l'utilisateur doit être très précis dans la façon dont il combine les mots-clés, alors que, sur papier, il ne prendrait même pas la peine d'écrire la preuve.

*Cette contribution est soutenue par un partenariat entre Nomadic Labs et l'Inria.

2 Automatisation par combinaison de transformations logiques

Nous proposons un prototype de tactique permettant de réconcilier les preuves Coq avec la logique des prouveurs du premier ordre. Par exemple, nous rendons possible une preuve complètement automatique de l'exemple ci-dessus. Cette tactique s'appuie sur une méthodologie qui consiste à :

1. Prouver automatiquement en Coq dans le contexte local des énoncés auxiliaires de premier ordre explicitant les propriétés des sous-termes (fonctions, inductifs, égalités d'ordre supérieur...) du but et des hypothèses en réduisant *in fine* ce dernier à un énoncé du premier ordre. Dans notre cas, nous implantons des transformations logiques indépendantes *certifiantes* qui utilisent les outils de métaprogrammation Ltac [2] et MetaCoq [4]. *Certifiant* signifie que chaque transformation prouve aussi la correction du résultat à chaque appel (donc au cours de son exécution, et non pas en amont).
2. Envoyer ce but du premier ordre et tous les lemmes auxiliaires à un prouveur du premier ordre externe. Si ce prouveur résout le but, un terme de preuve Coq est reconstitué (avant d'être *type-checké*) à partir du certificat produit par le prouveur.

Nous appliquons cette méthodologie en Coq dans la nouvelle tactique `snipe`. Les deux phases sont implantées comme suit :

1. Nous spécifions des transformations indépendantes et n'utilisant aucun axiome (1) prouvant les énoncés spécifiant que les constructeurs d'un type inductif sont injectifs et d'images directes disjointes (2) ajoutant (dans le contexte local) les définitions des fonctions qui apparaissent dans les hypothèses ou dans le but (3) transformant les égalités de la forme `f = g` entre deux objets d'ordre supérieur en des énoncés du premier ordre de la forme `forall (x1: A1)... (xn: An): f x1 ... xn = g x1 ... xn` (4) éliminant les points-fixes anonymes (opérateur `fix`) (5) éliminant les pattern-matching par analyse de cas (6) instanciant toutes les hypothèses (et éventuellement, lemmes auxiliaires) polymorphes par tous les sous-termes de type `Type` du but (monomorphisation). Ces transformations sont toutes combinées dans la première phase de la tactique `snipe`.
2. Ensuite, cette tactique fait appel au prouveur SMT veriT [1] comme *back-end* pour décharger le but du premier ordre, à l'aide du plugin SMTCoq [3], qui permet à Coq et des prouveurs SMT de communiquer. Il est important de remarquer que, pour cette étape, SMTCoq pourrait être remplacé par n'importe quelle tactique permettant de résoudre des buts du premier ordre.

La preuve de l'exemple ci-dessus devient tout simplement un appel à la tactique `snipe` (`Proof. snipe. Qed.`).

Le plugin `Sniper` présenté ici est disponible à l'adresse suivante : https://github.com/smtcoq/sniper. Dans notre exposé, nous détaillerons ses mécanismes, que nous avons esquissés ci-dessus.

3 Vers plus d'expressivité

Dans le futur, nous aimerions pouvoir appliquer `Sniper` à des buts plus expressifs, de façon à profiter davantage des heuristiques puissantes des prouveurs automatiques. Une première étape serait de traiter le cas de certains prédicats décidables, en les plongeant dans les booléens. À plus long terme, nous pourrons tenter d'instancier automatiquement des principes d'induction avec certains prédicats booléens, et les envoyer aux prouveurs externes.

Références

[1] Thomas Bouton, Diego Caminha Barbosa De Oliveira, David Déharbe, and Pascal Fontaine. verit : An open, trustable and efficient smt-solver. In Renate A. Schmidt, editor, *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*, pages 151–156. Springer, 2009.

[2] David Delahaye. A Tactic Language for the System Coq. In Michel Parigot and Andrei Voronkov, editors, *Logic for Programming and Automated Reasoning, 7th International Conference, LPAR 2000, Reunion Island, France, November 11-12, 2000, Proceedings*, volume 1955 of *Lecture Notes in Computer Science*, pages 85–95. Springer, 2000.

[3] Burak Ekici, Alain Mebsout, Cesare Tinelli, Chantal Keller, Guy Katz, Andrew Reynolds, and Clark W. Barrett. Smtcoq : A plug-in for integrating SMT solvers into coq. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 126–133. Springer, 2017.

[4] Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. The MetaCoq Project. *J. Autom. Reason.*, 64(5) :947–999, 2020.