Trakt : Uniformiser les types pour automatiser les preuves (démonstration)

Denis Cousineau, Enzo Crance, Assia Mahboubi

Trakt : Uniformiser les types pour automatiser les preuves

Denis Cousineau[1], Enzo Crance[1,2], and Assia Mahboubi[2]

[1] Mitsubishi Electric R&D Centre Europe (MERCE), Rennes, France
[2] Inria Rennes Bretagne Atlantique, Rennes, France

Résumé

Dans un assistant de preuve comme Coq, un même objet mathématique peut souvent être formalisé par différentes structures de données. Par exemple, le type `Z` des entiers binaires, dans la bibliothèque standard de Coq, représente les entiers relatifs tout comme le type `ssrint`, des entiers unaires, fourni par la bibliothèque MathComp. En pratique, cette situation familière en programmation est un frein à la preuve formelle automatique. Dans cet article, nous présentons `trakt`, un outil dont l'objectif est de faciliter l'accès des utilisateurs de Coq aux tactiques d'automatisation, pour la représentation des théories décidables de leur choix. Cet outil construit une formule auxiliaire à partir d'un but utilisateur, et une preuve que cette dernière implique ce but initial. La formule auxiliaire est conçue pour être adaptée aux outils de preuve automatique (`lia`, SMTCoq, etc). Cet outil est extensible, grâce à une API permettant à l'utilisateur de définir plusieurs natures de plongements dans un jeu de structures de données de référence. Le méta-langage Coq-Elpi, utilisé pour l'implémentation, fournit des facilités bienvenues pour la gestion des lieurs et la mise en œuvre des parcours de termes en jeu dans ces tactiques.

1 Introduction

L'automatisation est un critère majeur dans l'adoption des assistants de preuve, que ce soit pour des applications académiques ou industrielles. Par exemple, les utilisateurs de l'assistant de preuve Isabelle/HOL [1] bénéficient de tactiques d'automatisation puissantes, en particulier celles basées sur le modèle de *marteaux* [2]. En Coq, des procédures de décision variées sont disponibles. Certaines travaillent sur des théories décidables spécifiques, comme la tactique `lia` [3] pour l'arithmétique linéaire entière. D'autres sont conçues pour travailler sur une combinaison de théories, comme la bibliothèque SMTCoq [4]. L'objectif de cette dernière est de connecter Coq à des solveurs SMT, et d'apporter ainsi aux utilisateurs de Coq la puissance de preuve de ces outils optimisés. Un des défis de l'automatisation en Coq est de dompter l'expressivité de Gallina, le langage de Coq, et de rendre les outils de preuve automatique disponibles à toutes les structures de données pouvant représenter une même théorie décidable.

Pour répondre à ce problème, nous proposons un outil de pré-traitement des buts Coq, qui intervient avant les tactiques d'automatisation. Il reformule le but de manière certifiée afin d'en donner une forme canonisée, par exemple adaptée à la spécification d'entrée d'une tactique d'automatisation arbitraire. L'outil se veut simple d'utilisation et extensible, exposant une API pour que l'utilisateur décrive précisément la cible attendue de la traduction.

2 Limites des interfaces des tactiques d'automatisation

L'expressivité offerte par Coq permet aux utilisateurs de représenter un même concept par des types possiblement très différents. Par exemple, la bibliothèque standard de Coq elle-même propose plusieurs types pour représenter les entiers : naturel, relatifs, binaires, machine, etc. Un utilisateur peut aussi choisir d'introduire sa propre formalisation des entiers, et y associer de nouvelles opérations, propriétés, relations, et appareils de notations. Par ailleurs, les développeurs des outils de preuve formelle automatique peuvent également être amenés à faire

des choix sur la forme syntaxique des buts qu'ils souhaitent traiter. Cette double variabilité, à la fois des spécifications d'entrée des tactiques et des buts énoncés par les utilisateurs, peut entraver le bon fonctionnement des tactiques d'automatisation, qui ne parviennent pas toujours à traiter tous les buts Coq attendus, même ceux qui représentent a priori des formules de la théorie décidée par l'algorithme sous-jacent.

Afin d'augmenter la compatibilité des buts avec les procédures de décision, une solution est d'ajouter une phase de pré-traitement (un *proxy*) entre ces buts et les tactiques associées. Par exemple, la tactique `zify` [5] a pour objectif de faire converger une variété de buts arithmétiques et logiques vers une forme canonique de ces buts compréhensible par la tactique `lia`. La tactique `zify` traduit certaines représentations des entiers (`nat`, `positive`, etc) vers `Z`, la représentation des entiers de la tactique ciblée. La traduction est certifiée, la forme canonique étant prouvée équivalente au but d'origine. De plus, l'outil est extensible car il met à disposition des classes de types Coq pour que l'utilisateur puisse déclarer des plongements depuis d'autres types, ainsi que la traduction de symboles et relations associés. La tactique exploite ensuite toutes les instances déclarées. Une autre tactique de pré-traitement est `mczify` [6]. Il s'agit d'un jeu d'instances des classes de types de `zify` conçues spécialement pour traduire un maximum de symboles de la bibliothèque MathComp [7].

L'objectif de ce travail est de généraliser ces interfaces, de sorte qu'elles puissent être exploitées par des outils d'automatisation arbitraires. La bibliothèque SMTCoq dont les tactiques d'automatisation ciblent les théories SMT (arithmétique, égalité, vecteurs de bits, symboles non interprétés, etc) sert ici d'exemple et de motivation. Celle-ci implémente en effet une forme de pré-traitement des buts, mais relativement sommaire et peu extensible. À notre connaissance, il n'existe pas de proxy à la fois extensible *via* des déclarations utilisateur, et générique quant aux tactiques d'automatisation ciblées.

3 Un proxy générique et extensible

L'interface de l'outil que nous proposons, `trakt`[1], permet d'effectuer des déclarations à la `zify`, mais pour des types cibles arbitraires. Par ailleurs, cette interface est complètement générique : elle n'est liée à aucune bibliothèque spécifique et ne cible aucune tactique d'automatisation en particulier. L'outil `trakt` propose ainsi des commandes Coq permettant à l'utilisateur de déclarer facilement des plongements entre types, des symboles connus ou bien des relations, alimentant une base de données. Celle-ci est ensuite exploitée par une tactique de traduction, `trakt`, dont le comportement est paramétré par l'utilisateur grâce à ses déclarations. La traduction est ensuite entièrement automatique. À partir d'un but Coq `G`, elle génère à la fois un nouveau but `G'`, la forme canonisée de `G`, et un terme de preuve de type `G' -> G`.

Cet outil est écrit en Coq-Elpi [8], un méta-langage pour Coq dans le paradigme de la programmation logique. Ce choix de méta-langage s'est révélé pertinent à plusieurs titres. Par exemple, l'encodage des termes Coq en HOAS (*Higher Order Abstract Syntax*) [9] permet de traiter les lieurs sans devoir gérer des indices de De Bruijn [10], et de créer temporairement de nouveaux constructeurs de termes Coq, afin d'annoter librement des nœuds. L'association des variables d'unification de Coq à des variables Prolog permet de forger facilement des termes à trous. Enfin, le niveau d'abstraction offert permet d'éviter d'expliciter tous les détails de l'arbre de syntaxe abstraite lors du traitement d'un terme.

Notre prototype a été testé avec succès sur des exemples de buts variés. Lors de l'exposé, nous en ferons une démonstration en insistant sur la facilité de prise en main pour l'utilisateur et l'augmentation effective du niveau d'automatisation dans Coq.

1. "entonnoir" en norvégien

Références

[1] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. *Isabelle/HOL : a proof assistant for higher-order logic*, volume 2283. Springer Science & Business Media, 2002.

[2] Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C Paulson, and Josef Urban. Hammering towards QED. *Journal of Formalized Reasoning*, 9(1) :101–148, 2016.

[3] Frédéric Besson. Fast reflexive arithmetic tactics the linear case and beyond. In *International Workshop on Types for Proofs and Programs*, pages 48–62. Springer, 2006.

[4] Burak Ekici, Alain Mebsout, Cesare Tinelli, Chantal Keller, Guy Katz, Andrew Reynolds, and Clark Barrett. SMTCoq : A plug-in for integrating SMT solvers into Coq. In *International Conference on Computer Aided Verification*, pages 126–133. Springer, 2017.

[5] Frédéric Besson. ppsimpl : a reflexive Coq tactic for canonising goals, 2017.

[6] Kazuhiko Sakaguchi. `mczify`. https://github.com/math-comp/mczify.

[7] Assia Mahboubi and Enrico Tassi. Mathematical components, 2017.

[8] Enrico Tassi. Elpi : an extension language for Coq (Metaprogramming Coq in the Elpi λProlog dialect). 2018.

[9] Frank Pfenning and Conal Elliott. Higher-order abstract syntax. *ACM sigplan notices*, 23(7) :199–208, 1988.

[10] Nicolaas Govert De Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. In *Indagationes Mathematicae (Proceedings)*, volume 75, pages 381–392. Elsevier, 1972.