Rapport

Maryem Hajji, Léa Riant, Ryan Lahfa, Ivan Hasenohr

Table des matières

Introduction

Avant d'expliquer en quoi consiste un assistant de preuve, donnons quelques éléments d'histoire autour de ces derniers.

Courte histoire des assistants de preuve et du rêve d'Hilbert

En août 1900, David Hilbert présente ses 23 problèmes, dont le second est la cohérence de l'arithmétique, fracassé par le résultat d'incomplétude de Gödel (qui ne résoud pas tout à fait la question) en 1931, et dont une réponse positive est obtenue par Gantzen à l'aide de la récurrence transfinie. C'est l'élan qui va lancer la théorie de la démonstration.

En 1966, de Bruijn lance le projet Automath qui a pour visée de pouvoir exprimer des théories mathématiques complètes, c'est-à-dire des théories qui sont des ensembles maximaux cohérents de propositions, i.e. le théorème d'incomplétude de Gödel ne s'y applique pas notamment.

Peu après, les projets Mizar, HOL-Isabelle et Coq naissent pour devenir les assistants de preuve mathématiques que l'on connaît.

Principe d'un assistant de preuves

Ces projets mettent à disposition un ensemble d'outil afin d'aider le mathématicien à formaliser sa preuve dans une théorie mathématiques de son choix: ZFC, la théorie des types dépendants, la théorie des types homotopiques par exemple.

Certains assistants de preuve ne se contentent pas de vérifier la formalisation d'une preuve mais peuvent aussi effectuer de la décision (dans l'arithmétique de Presburger par exemple).

Enjeu d'un assistant de preuves et exemples d'usages

L'enjeu des assistants de preuve et des concepts utilisés derrière dépasse le simple outil de mathématicien.

D'une part, ils permettent d'attaquer des problèmes qui ont résisté pendant longtemps, le théorème des quatre couleurs par exemple.

D'autre part, leurs usages se généralisent afin de pouvoir faire de la certification informatique, démontrer

qu'un programme vérifie un certain nombre d'invariants, par exemple, dans l'aviation, des outils similaires sont employés pour certifier le comportement de certaines pièces embarquées.

Éléments de théorie des assistants de preuves

Nous nous attacherons pas à faire un état du fonctionnement des assistants de preuves, ceux là dépassent largement le cadre d'une licence, mais on peut donner quelques éléments d'explications.

Distinguons deux opérations, celle de la vérification de preuve et celle de la déduction automatique.

Notons que dans un premier temps, la plupart des opérations idéales d'un assistant de preuve sont indécidables, c'est-à-dire, qu'il n'existe pas d'algorithme permettant de calculer le résultat en temps fini.

Dans ce cas, afin de pouvoir vérifier une preuve, il faut l'écrire dans un langage où toutes les étapes sont des fonctions récursives primitives (ou des programmes), ce qui les rend décidables par un algorithme. L'enjeu ensuite est de le faire efficacement, bien sûr.

Ainsi, rentre en jeu les notions de mots, de langages, de confluences et de systèmes de réécritures et d'avoir des algorithmes de bonne complexité temporelle et mémoire afin de pouvoir manipuler les représentations internes d'une preuve et décider s'ils sont des preuves du résultat désiré.

Au dessus de cela, on a besoin de se donner des théories axiomatiques dans lequel on travaille, par exemple ZFC, Peano, la théorie des catégories, la théorie des types dépendants, la théorie des types homotopiques. Dans notre cas, Lean utilise la théorie des types dépendants par défaut mais propose la version homotopique aussi, qui est plus délicate à manipuler. De cela, on peut construire des notions d'ensembles, d'entiers naturels, de catégories aussi.

Ceci est pour la partie vérification et fondations théoriques du modèle.

Pour la partie automatique, selon la logique, le problème passe d'indécidable à décidable, par exemple, pour le calcul des propositions, le problème est décidable mais de classe de complexité co-NP-complete (le complémentaire de la classe NP-complete), indiquant que les algorithmes de décisions prennent un temps exponentiel certainement. En somme, c'est un problème très difficile, mais sur lequel il a été possible d'avoir des résultats positifs, notamment un qui a résolu un problème de longue date sur lequel aucune bille n'était disponible: la conjecture de Robbins, 1933, résolue en 1996 avec un assistant de preuve à déduction automatique EQP.

Dans une certaine mesure, Lean est capable d'assister à trouver des morceaux de preuve par lui-même à l'aide de tactiques qui peuvent être aussi écrite par les utilisateurs afin d'améliorer l'intelligence de Lean dans certains contextes (chasse aux diagrammes en catégories par exemple).

Détail des exercices du « Number Games » de Kevin Buzzard

Tactiques

— On suppose dans cette partie que:
- Σ est un alphabet fini qui contient les lettres de l'alphabet latin, les parenthèses et les opérateurs arithmétiques.
- Σ^* est l'ensemble des mots possibles qu'on peut construire à partir de Σ.
- F, A et B sont des mots de Σ^*.

— **refl**: Cette tactique correspond à la réflexivité de l'égalité, d'où le nom **refl**. Elle peut s'appliquer pour prouver toute égalité de la forme $A = A$. C'est à dire, toute égalité dont les deux membres sont égaux terme à terme.
Exemple: soient x, y, z, w des entiers naturels, alors on peut prouver que $x + y * (z + w) = x+y*(z+w)$ en exécutant l'instruction $\{\textbf{refl,}\}$.

— **rw**: Le nom de cette tactique (rw) correspond au mot anglais *rewrite*. Elle s'applique dans 2 cas distincts:

Soit H une hypothèse, sous la forme $A = B$. Supposons que l'équation à démontrer est le mot F.

Si F contient au moins un A, l'instruction $\{$**rw** H**,**$\}$ dérive un mot F' du mot F, en effectuant un seul changement: tous les As (présents dans F) sont réécrits en Bs. De même, si F contient au moins un B et si on utilise $\{$**rw** $\leftarrow$ **H,**$\}$, alors le seul changement sera: tous les Bs (présents dans F) sont réécrits en As.

Soit $T : A = B$, c'est à dire T est une preuve de $A = B$, supposé faite à un niveau qui précède le niveau traité. Dans ce cas, elle figure sur le menu des théorèmes. Alors $\{$**rw T,**$\}$ (respectivement $\{$**rw** $\leftarrow$ **T,**$\}$) dérive un mot F' du mot F, en effectuant un seul changement: tous les As (resp. Bs) sont remplacés par des Bs (resp. As).

— **simp**: C'est une tactique de haut niveau. Elle est disponible à partir du dernier niveau de *Addition World*. Son principe est le suivant: elle utilise la tactique **rw** avec les preuves des théorèmes d'associativité et de commutativité de l'addition pour prouver une certaine égalité (les preuves de l' associativité et la commutativité de la multiplication sont disponibles à partir du dernier niveau de *Multiplication World*). De plus, à l'aide du langage de métaprogrammation de Lean, on peut éventuellement apprendre à **simp** à simplifier une variété de formules plus large en utilisant d'autres preuves outre celles de l'associativité et de la commutativité.
Exemple: Soient x, y, z, w, u des entiers naturels, alors on peut démontrer que $x + y + z + w + u = y + (z + x + u) + w$ en utilisant $\{$**simp,** $\}$

Addition World

Addition World est le premier monde de **Natural Number Game**. Dans ce monde, on dispose principalement de 3 tactiques: *refl*, *rw* (dont l'application était initiée dans *Tutorial*) et *induction*.
En plus, chaque théorème, une fois démontré, sera utilisé comme un résultat acquis dans les démonstrations de tous les théorèmes qui suivent. Par exemple, en commençant *Addition World*, on peut utiliser les deux théorèmes suivants: add_zero et add_succ, qui sont supposés démontrés dans la partie *Tutorial*.
Addition World contient 6 niveaux: zero_add, add_assoc, succ_add, add_comm, succ_eq_add_one et add_right_comm. Détaillons la démonstration du théorème suivant:
Le 5ème niveau : succ_eq_add_one

pour tout entier naturel n, $succ(n) = n + 1$

Preuve **rw one_eq_succ_zero,** : c'est plus facile de manipuler le chiffre 0 que le chiffre 1. On réécrit donc 1 en succ(0), puisque $1 = succ(0)$ (la preuve de cette égalité est one_eq_succ_zero). On obtient $succ(n) = n + succ(0)$
rw add_succ, : add_succ fournit l'égalité $n + succ(0) = succ(n + 0)$, on l'utilise alors pour réécrire $succ(n) = n + succ(0)$ en $succ(n) = succ(n + 0)$. Ainsi, on pourra utiliser un des théorèmes qui manipulent le chiffre 0
rw add_zero, : utilisation de ce théorème pour réécrire $n + 0$ en n
refl,

Multiplication World

Dans ce monde, les théorèmes reposent principalement sur les propriétés basiques de la multiplication, tels que la commutativité, l'associativité, et la distributivité de la multiplication par rapport à l'addition dans les deux sens (à gauche et à droite). *Multiplication World* contient 9 niveaux: zero_mul, mul_one, one_mul, mul_add, mul_assoc, succ_mul, add_mul, mul_comm et mul_left_comm.
Nous explicitons la démonstration du théorème suivant:
Le 4ème niveau : mul_add

La multiplication est distributive, c'est à dire pour tous entiers naturels a, b et t :

$$t * (a + b) = t * a + t * b$$

Preuve **induction a with d hd,** : Dans l'induction, a est renommé en d qui varie inductivement et hd est l'hypothèse d'induction sur d (cas de base: $d = 0$, cas d'induction: on suppose hd, on démontre $h(succ(d))$)

Cas de base: montrons que $t*(0+b) = t*0+t*b$
rw zero_add, : on remplace $0+b$ par b, on obtient $t*b = t*0+t*b$
rw mul_zero, : on remplace $t*0$ par 0, on obtient $t*b = 0+t*b$
rw zero_add, : on obtient $t*b = t*b$
refl,
Cas d'induction: supposons hd : $t*(d+b) = t*d+t*b$ *et montrons* $h(succ(d)) : t*(succ(d)+b) = t*succ(d)+t*b$
rw succ_add, :une solution serait de se ramener à une équation où l'un des deux membres est égal à un membre de *hd*. Pour faire cela, on utilise succ_add qui s'applique uniquement sur une quantité de la forme $succ(d)+b$ (d et b étant deux entiers naturels quelconques), nous permettant ainsi de la remplacer par $succ(d+b)$
rw mul_succ, : on utilise *mul_succ* (a b : mynat) : $a*succ(b) = a*b+a$
rw hd, on remplace $t*(d+b)+t$ par $t*d+t*b+t$ en utilisant *hd*, on obtient $t*d+t*b+t = t*succ(d)+t*b$
rw add_right_comm, : on applique la commutativité de l'addition pour remplacer $t*b+t$ par $t+t*b$
rw ← mul_succ, : on utilise rw ← pour remplacer $t*d+t$ (qui est le membre droit de l'égalité qui correspond au théorème mul_succ) par $t*succ(d)$, on obtient $t*succ(d)+t*b = t*succ(d)+t*b$
refl,

Power World

Ce monde contient 8 niveaux: zero_pow_zero, zero_pow_succ, pow_one, one_pow, pow_add, mul_pow, pow_pow et add_squared.
Nous explicitons la démonstration du théorème suivant:
Le 7ème niveau: add_squared (Cas particulier de la formule du binôme de Newton: $(a+b)^n = \sum_{k=0}^{n} \frac{n!}{k!(n-k)!} a^n b^{n-k}$, pour $n=2$)

pour tous entiers naturels a et b :
$(a+b)^2 = a^2+b^2+2*a*b$

Preuve *On simplifie les puissances, en réécrivant les puissances 2 en fonction de 0*
rw two_eq_succ_one, : on utilise la preuve de $succ(1) = 2$ pour réécrire le chiffre 2 en $succ(1)$
rw one_eq_succ_zero, : on réécrit 1 en $succ(0)$, on obtient donc $(a+b)^{succ(succ(0))} = a^{succ(succ(0))} + b^{succ(succ(0))} + succ(succ(0))*a*b$
repeat rw pow_succ, : on obtient $(a+b)^0*(a+b)*(a+b) = a^0*a*a+b^0*b*b+succ(succ(0))*a*b$
repeat rw pow_zero, : on obtient $1*(a+b)*(a+b) = 1*a*a+1*b*b+succ(succ(0))*a*b$
simp, : on obtient $(a+b)*(a+b) = a*a+(b*b+a*(b*succ(succ(0))))$, donc simp, dans ce cas, applique le théorème one_mul(m : mynat) : $m*1 = m$
repeat rw mul_succ, : on obtient $(a+b)*(a+b) = a*a+(b*b+a*(b*0+b+b))$
simp, : on obtient $(a+b)*(a+b) = a*a+(b*b+a*(b+b))$, donc simp, dans ce cas, applique les théorèmes mul_zero(a : mynat):$a*0 = 0$ et zero_add(n : mynat):$0+n = n$
*On développe (a + b) * (a + b) :*
rw mul_add,
On développe $(a+b)*a$ *:*
rw mul_comm,
rw mul_add,
On développe $(a+b)*b$ *:*
rw mul_comm (a + b) b,
rw mul_add,
simp, *On met les termes du membre de gauche dans le bon ordre*
rw ← add_assoc (a * b) (a * b) (b * b), : on obtient $a*a+(a*b+a*b+b*b) = a*a+(b*b+a*(b+b))$
rw add_right_comm,
rw add_comm (a * b) (b * b),
rw add_assoc (b * b) (a * b) (a * b), : on obtient $a*a+(b*b+(a*b+a*b)) = a*a+(b*b+a*(b+b))$
On factorise par a :
rw ← mul_add a b b, : on obtient $a*a+(b*b+a*(b+b)) = a*a+(b*b+a*(b+b))$
refl,

Excursion dans le formalisme des espaces métriques