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

Exact, Intro, Have, Apply Ici nous allons présenter 4 techniques fondamentales pour l'utilisation de fonctions, une fonction f : $A \to B$ pour A et B deux types étant simplement un élément de type $A \to B$, qui à une preuve de A renvoie une preuve de B.

Exact

La première de ces tactiques est *exact*. Elle permet de dire à Lean que le but recherché correspond exactement à ce que vous lui indiquez. Par exemple, si le but est $\exists p$ de type P, et que vous disposez de p de type P, alors *exact p,* terminera la preuve. De même, si le but est $\exists q$ de type Q et que vous disposez d'un élément p de type P et d'une fonction f : $P \to Q$, alors *exact f(p),* terminera la preuve.

Intro

Lorsque vous manipulez des fonctions, Lean peut vous demander de créer une fonction d'un type P vers un type Q. Une méthode est alors de d'émettre l'hypothèse qu'on dispose d'un p de type P à partir duquel vous fabriquerez un élément de Q. *intro p,* fait cela : vous disposerez alors d'une preuve p de P et votre but sera reformulé en Q.

De façon similaire, lorsque de le but est de la forme $P_1 \to P_2 \to ... \to P_n \to Q$, *intros* $p_1 p_2 ... p_n$ change le but en Q.

Have

Cette technique permet de renommer des variables :

par exemple, si vous disposez de p de type P et de f : $P \to Q$, alors *have q : Q := f(p),* vous permet de renommer un élément $q = f(p)$. Le principe du Démiurge nous permet en effet de renommer comme on veut ce que l'on veut, ce qui garantit la validité de la preuve dans le cas de l'utilisation de have.

Apply

Cette technique vous permet de modifier le but sans rajouter de variables : de fait, elle raisonne comme ceci : vous avez pour but un élément de Q. Or vous disposez d'une fonction f : $P \to Q$. De ce fait, pour disposer d'un élément de Q, il vous suffit de disposer d'un élément de P, car f(p) sera dans Q. *apply f,* fait exactement ça, et donc changera le but de Q en P.

IV : Function World Ce monde nous introduit un outil fondamental de Lean : les fonctions.
Un élément important à remarquer est qu'en Lean, toutes les fonctions sont curryfiées.

Voici un exemple de niveau de ce monde, le niveau 6, qui demande de créer une fonction de fonctions assez fastidieuse, et qui utilise le fait que ces fonctions sont curryfiées.
L'énoncé se formule comme ceci :
$(PQR : Type) : (P \to (Q \to R)) \to ((P \to Q) \to (P \to R))$
La preuve est de fait assez simple :
intros f g p, – On introduit les différents éléments/fonctions pour créer la fonction demandée
apply f p, – On modifie le but à l'aide de la fonction curryfiée
exact g p, – On trouve le résultat demandé
Ce qui conclut la preuve.

V : Proposition World Dans ce monde on aborde un aspect fondamental de l'assistant de preuves Lean : une preuve est composée d'implications, et c'est ici que les fonctions prennent toute leur importance : pour montrer que A implique B, il suffit de créer une fonction de A vers B, soit un élément de type $A \to B$.

Pour illustrer ce point, voici un exemple simple, le tout premier niveau de Proposition World.

Lemme : if P is true and $P \to Q$ is true, then Q is true.

Soit en Lean : *(P Q : Prop) (p : P) (h : P → Q) : Q*
Donc, en français, on dispose d'une preuve de P, et d'une fonction de P dans Q (i-e d'un élément de type $P \to Q$), trouvons un élément de type Q (montrons que Q est vrai).
Ce qui se résout tout aussi succintement : *exact h(p),*.

Un autre niveau intéressant est le niveau 8, qui propose une preuve du lemme suivant, une implication de l'équivalence entre une proposition et sa contraposée (si cela a du sens) :

Lemme : $(P \to Q) \to (\neg Q \to \neg P)$.

En Lean, on demande donc de créer une fonction qui prend une preuve que $P \to Q$ et renvoie une preuve de $\neg Q \to \neg P$.
Pour cela, la première étape est de disposer d'une preuve de $P \to Q$:
intro f,
Lean nous demande alors de créer un élément de type $\neg Q \to \neg P$, qui serait l'image de la fonction qu'il nous est demandé de créer.
L'astuce est ensuite de revenir à la définition de $\neg P$: $\neg P \equiv P \to false$. On retranscrit cette définition :
repeat{rw not_iff_imp_false},
Le but est alors réécrit en $(Q \to false) \to P \to false$, ce qui revient à créer une fonction curryfiée des éléments de type $(Q \to false) \times P$ vers les preuves de $false$
On réapplique la même technique d'introduire un élément de chacun des ensembles de départ :
intros h p,
On dispose alors d'un élément p de P, d'une fonction f de P dans Q et d'une fonction h de Q dans $false$, et il nous faut créer une preuve de $false$, qui est facilement trouvable avec :
exact h(f(p)),
Ce qui conclut la preuve.

VI : Advanced Proposition World Dans ce monde on démontre à l'aide de fonctions et de nouvelles méthodes les règles de base de la manipulation de consjonctions et disjonctions logiques. Un exemple combinant la plupart des nouvelles méthodes est le Lemme suivant :

Lemme : Soient P,Q et R trois propositions.
ALors $P \wedge (Q \vee R) \iff (P \wedge Q) \vee (P \wedge R)$.

Ici on ne démontrera que l'implication directe, l'implication réciproque se faisant de façon similaire.
Pour séparer les implications, une technique existe : *split*, qui permet de montrer d'abord l'implication directe puis l'implication réciproque. A noter que cette technique permet aussi de séparer le but en plusieurs buts lorsqu'on a à montrer une conjonction de propositions.
Pour gérer les disjonctions de propositions, la technique *cases* existe et permet, par exemple, quand on sait que $P \vee Q$, dans un premier temps supposer P puis supposer Q. Cette technique permet aussi de séparer les conjonctions connues en plusieurs nouvelles données : si l'on a un élément pq de $P \wedge Q$, *cases pq with p q* nous renvoie deux éléments p et q de P et Q respectivement.
Finalement, lorsqu'on doit montrer une disjonction de propositions, il suffit d'en montrer une, et les techniques *left* et *right* nous permettent de choisir la proposition à démontrer.
La preuve est donc la suivante :
intro h, – h de type $P \wedge (Q \vee R)$
cases h with p qor, – p de type P , qor de type $Q \vee R$
cases qor with q r, – On sépare en deux cas en fonction de la disjonction :

– Premier cas q de type Q
left, – On choisit de montrer $P \wedge Q$
split, – On sépare en deux buts
exact p,
exact q,

– Deuxième cas : r de type R
right, – On choisit de montrer $/P \wedge R$
split, – On sépare en deux buts
exact p,
exact r,
Ce qui conclut la preuve de l'implication directe.

Excursion dans le formalisme des espaces métriques