Traduction prouvée de HOL4 vers le premier ordre

Thibaut Gauthier[1] & Chantal Keller[2] & Michael Norrish[1]

[1] *Canberra Research Lab., NICTA, Australie &* [2] *Aarhus University, Danemark*
<thibault_gauthier@hotmail.fr> <ckeller@cs.au.dk> <Michael.Norrish@nicta.com.au>

Résumé

Nous présentons une traduction de la logique d'ordre supérieur polymorphe vers la logique du premier ordre monomorphe implantée et prouvée correcte dans l'assistant de preuves HOL4, faisant notamment intervenir de nouvelles techniques de monomorphisation. Elle permet d'interfacer HOL4 avec des prouveurs du premier ordre en vue de décharger certains buts à ces prouveurs automatiques, tout en ayant confiance dans la traduction. Des expériences utilisant le prouveur du premier ordre avec arithmétique Beagle montrent son expressivité (combinaison de raisonnements propositionnel et arithmétique) et l'efficacité de notre monomorphisation.

1. Introduction

Pour bénéficier de l'état de l'art en matière d'automatisation, de nombreux assistants de preuve font appel à des prouveurs automatiques, qu'ils soient internes [9, 12] ou externes [1, 11], afin de résoudre certains de leurs buts. Si un prouveur interne confère une autonomie à l'assistant de preuves dans lequel il est implanté et peut donc offrir des garanties de complétude – on peut établir formellement que le prouveur sera capable de prouver tout ce qui est prouvable – l'utilisation de prouveurs externes est souvent plus simple – car il suffit de vérifier ses réponses *a posteriori* et non de montrer la correction complète du prouveur – plus efficace – car il peut être très optimisé sans que cela complique la vérification de ses réponses – et également plus générique – on peut définir une interface entre assistants de preuves et prouveurs automatiques indépendante d'un pouveur particulier. Les deux manières de procéder sont généralement complémentaires : les prouveurs internes sont utilisés afin de vérifier les traces générées par les prouveurs externes.

Nous présentons ici une traduction de la logique de l'assistant de preuve HOL4 vers la logique du premier ordre avec arithmétique (codée dans le format TFF), destinée à pouvoir décharger certains buts de HOL4 vers des prouveurs automatiques du premier ordre. Afin de ne pas compromettre la cohérence de HOL4 :

- en amont, la traduction est entièrement prouvée correcte en HOL4 ;
- en aval, un premier vérificateur permet de rejouer les preuves éventuellement fournies par le prouveurs automatique (au format TFF) en HOL4 à l'aide de ses prouveurs internes, notamment metis [9] pour la logique du premier ordre et Cooper pour l'arithmétique.

L'utilisation du format TFF générique rend la traduction et la vérification de preuves indépendantes du prouveur externe utilisé, et offre donc la possibilité d'en utiliser plusieurs en parallèle ou en fonction des besoins.

Cette traduction s'appuie sur les traductions déjà existantes, notamment celles implantées pour la tactique sledgehammer [11] en Isabelle/HOL, mais présente également de nouveaux concepts, notamment en ce qui concerne la *monomorphisation* (c'est-à-dire la suppression de la quantification sur les variables de type), où nous avons cherché à donner un algorithme simple mais néanmoins efficace. Elle permet également de gérer à la fois les quantificateurs du premier ordre et l'arithmétique.

Cette traduction générique est utilisée pour implanter la nouvelle tactique BEAGLE_TAC (présentée en FIGURE 1), faisant appel au récent prouveur du premier ordre avec arithmétique

Beagle [2] (sans vérification de preuves dans la version actuelle). Bien qu'offrant moins de garanties de correction que METIS_TAC– la tactique faisant appel au prouveur du premier ordre interne à HOL4– et étant pour l'instant moins rapide, cette première version de BEAGLE_TAC est plus expressive car elle combine raisonnement propositionnel avec arithmétique. Il s'agit ici d'une des premières utilisations de Beagle, fournissant ainsi un ensemble de tests conséquent.

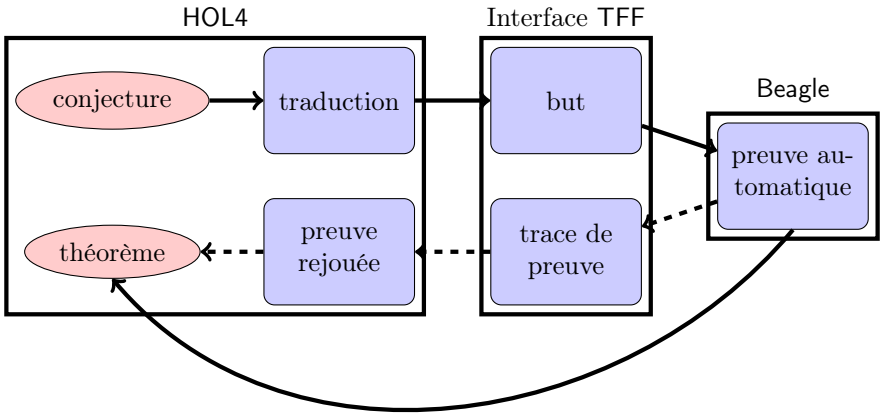


FIGURE 1 – La tactique BEAGLE_TAC : la version courante utilise Beagle comme un oracle (flèches pleines), mais ses traces peuvent être rejouées à l'aide de METIS_TAC et COOPER_TAC (flèches pointillées).

Le code présenté dans cet article est disponible à l'adresse `https://github.com/barakeel/HOLtoTFF` .

1.1. Assistants de preuve et prouveurs automatiques

Les assistants de preuve sont des logiciels basés sur un petit noyau implantant un vérificateur pour une logique très expressive, permettant d'énoncer et de prouver de manière très sûre des théorèmes dans une grande variété de domaines. Au-dessus de ce petit noyau, une interface utilisateur contenant notamment un ensemble de tactiques permet d'établir les preuves pas à pas. La grande expressivité de la logique implantée par ces prouveurs limite l'automatisation possible pour la recherche de preuve, demandant ainsi une assez grande expertise pour être utilisés.

A contrario, les prouveurs automatiques implantent une logique moins expressive, mais pour laquelle une recherche de preuve – bien qu'incomplète – est possible. Ils demandent donc moins de travail à l'utilisateur, mais offrent très peu de garanties de sûreté.

La TABLE 1 résument leurs points forts et faibles, soulignant leur complémentarité : une interaction entre les deux permettrait d'offrir à la fois l'automatisation lorsqu'elle est possible et la sûreté.

Dans cet article, nous nous intéressons plus particulièrement aux assistants de preuve basés sur la logique d'ordre supérieur, notamment HOL4, et aux prouveurs du premier ordre avec théories, comme Beagle. La logique implantée par les deuxièmes étant un sous-ensemble de celle implantée par les premiers, elle définit une interface commune leur permettant de communiquer.

	Assistants de preuve	Prouveurs automatiques
Prouveurs	HOL4, Isabelle/HOL, HOL Light, Coq, ...	Beagle, SPASS, Vampire, Z3, veriT ...
Logique	Ordre supérieur ou Théorie des types	Premier ordre et combinaison de théories
Automatisation	Guidage fastidieux de l'utilisateur	Preuve entièrement automatisée
Sûreté	Basé sur un petit noyau vérifiable	Code très grand difficile à prouver, mais retourne parfois des traces vérifiables *a posteriori*

TABLE 1 – Différences entre assistants de preuve et prouveurs automatiques

1.2. Interface : le format TFF

Cette interface est décrite par le format générique TFF [1] (signifiant "Typed First-Order Form"), promu par TPTP [2] (signifiant "Thousands of Problems for Theorem Provers") regroupant une grande base de données de problèmes pour prouveurs du premier ordre. Il permet de décrire tout problème du premier ordre avec arithmétique (entière, rationnelle ou réelle). Il propose également un format générique pour les traces de preuves [3].

L'utilisation d'un format générique permet de ne pas dépendre de prouveurs en particuliers, comme l'est la traduction décrite dans la partie 2.

1.3. Plan

Le papier est organisé comme suit. La partie 2 présente une traduction de la logique d'ordre supérieur vers la logique du premier ordre, entièrement implantée et prouvée en HOL4. Elle présente de nouveaux aspects, notamment concernant la monomorphisation (partie 2.2). Cette traduction est utilisée pour définir la nouvelle tactique BEAGLE_TAC (partie 3.2), dont l'efficacité est ensuite testée (partie 3.3). La partie 4 décrit comment, dans un deuxième temps, vérifier les traces au format TFF générées par certains prouveurs (dont Beagle). Enfin, nous discutons des travaux similaires et futurs (partie 5) avant de conclure.

2. Une traduction prouvée de la logique d'ordre supérieur vers la logique du premier ordre

Dans cette partie, nous expliquons notre traduction prouvée des buts HOL4 vers le format TFF, c'est-à-dire la logique du premier ordre avec arithmétique. Cela se fait par étapes de différentes sortes :

- des étapes effectivement de traduction de l'ordre supérieur vers le premier ordre (notamment la monomorphisation, le λ-lifting et la suppression des fonctions passées en arguments d'autres fonctions) ;
- d'autres étapes intermédiaires pour tenir compte du fait que l'on veut donner le but à un prouveur du premier ordre : il résout un problème de satisfiabilité (et non de prouvabilité),

1. Le format TFF est décrit à l'adresse `http://www.cs.miami.edu/~tptp/TPTP/TR/TPTPTR.shtml#TypeSystem`.
2. La bibliothèque TPTP est disponible à l'adresse `http://www.cs.miami.edu/~tptp`.
3. Le standard des preuves écrites au format TFF est présenté à l'adresse `http://www.cs.miami.edu/~tptp/TPTP/QuickGuide/Derivations.htm`.

attendu en forme normale conjonctive présentée sous forme d'une liste de clauses, et faisant intervenir certains types prédéfinis.

Nous présentons dans un premier temps les différentes étapes de la traduction, avant de détailler notre algorithme utilisé pour l'une d'entre elles : la monomorphisation.

2.1. Les étapes de la traduction

La traduction est une succession de petites étapes, chacune étant prouvée correcte. Nous présentons ici l'ordre de ces étapes, en précisant quel outil a été utilisé pour les implanter. Les étapes concernant réellement le passage au premier ordre sont notées en gras.

1. **Monomorphisation** : Il est souvent admis que pour des raisons d'efficacité, il est préférable d'effectuer cet étape en premier. La partie 2.2 décrit notre algorithme de monomorphisation.
2. Négation de la conclusion : la prouvabilité du but est équivalente à la non-satisfiabilité de la formule donnée au prouveur.
3. Mise sous forme normale conjonctive : cette étape est réalisée par la fonction CNF_CONV implantée en HOL4 par Joe Hurd en octobre 2001.
4. **λ-lifting** : cette étape consiste à éliminer les variables libres apparaissant dans des définitions locale. Elle est implantée de la même manière que dans [6]. Cette étape auraient pu être remplacés par l'utilisation de combinateurs [8]. Ces deux approches ont été testés dans le développement de sledgehammer [13].
5. Élimination des arguments booléens qui ne sont pas des variables : le format TFF propose un type $\$o$ pour désigner le type de retour des prédicats. Il ne peut être utilisé pour les arguments booléens des fonctions ; leur type est donc remplacé par un nouveau type `bool` [13].
6. Mise sous forme d'un ensemble de clauses par élimination des quantificateurs existentiels et distribution des quantificateurs universels par rapport au conjonctions.
7. **Défonctionalisation** : cette étape consiste à éliminer les fonctions passées en arguments à d'autres fonctions [8, 13]. Les fonctions arithmétiques sont néanmoins conservées, puisqu'elles font partie de notre langage cible. Elle est réalisée en fin de traduction afin de limiter son application : par exemple, l'élimination de l'ordre supérieur dans la formule $\exists f.\ \forall x.\ f\ x = 0$ aura déjà été réalisée par les étapes précédentes.
8. Case-split des variables booléennes liées.
9. Correspondance entre types entiers : le type HOL4 `num` représentant les entiers positifs est déplié en un entier `int` et la propriété qu'il est positif.

Chaque étape étant prouvée correcte en HOL4, il en résulte qu'une preuve d'un problème traduit constitue également une preuve du problème originel.

2.2. Algorithme de monomorphisation

Cette étape consiste à instancier les variables de type apparaissant dans les théorèmes prouvés en amont par plusieurs types concrets, le format TFF n'ayant pas de polymorphisme. La difficulté est de trouver les types concrets avec lesquels instancier ces variables tout en conservant la prouvabilité ; cette étape est incomplète [5], mais des heuristiques donnent de bonnes performances [4].

La correction de la monomorphisation est simple à établir, puisqu'il s'agit d'une ou plusieurs instanciations de variables de type (implicitement) universellement quantifiées.

Cette partie présente l'heuristique développée pour la traduction de HOL4 vers TFF. Nous présentons d'abord le problème sur un exemple (qui servira dans toute la suite de cette partie), avant d'expliquer notre algorithme de monomorphisation. Ses performances seront évaluées dans la partie 3.

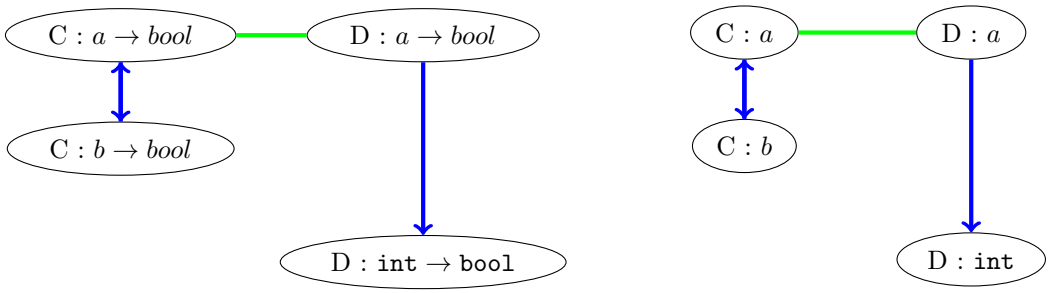


FIGURE 2 – Graphe de dépendance de l'exemple de la partie 2.2.1 (à gauche) et un graphe équivalent (à droite)

2.2.1. Exemple

Supposons prouvés les théorèmes $\forall x : a.\ C\ x \Rightarrow D\ x$ et $\forall x : b.\ C\ x$ où a et b sont des variables de type (implicitement quantifée universellement). On cherche à démontrer le but $D\ 42$. Les instanciations cherchées sont la substitution $\{a \mapsto \texttt{int}\}$ pour le premier théorème et la substitution $\{b \mapsto \texttt{int}\}$ pour le second théorème.

Nous illustrerons sur cet exemple les étapes de monomorphisation décrites dans la suite de cette partie.

2.2.2. Graphe de dépendance

Cette partie présente de manière théorique notre algorithme de monomorphisation.

Étant donné un problème, nous définissons un graphe de dépendance comme suit.

Définition. (Graphe de dépendance) Un graphe de dépendance est constitué de nœuds et de deux types d'arêtes.

- Les nœuds sont les constantes présentes dans la liste de théorèmes et dans le but, associées à leur type. Deux mêmes constantes sont représentées par des nœuds différents si elles proviennent de deux propositions (théorèmes ou buts) différentes.
- Les arêtes de substitution sont orientées et relient deux constantes vérifiant les propriétés suivantes :
  - la constante origine n'appartient pas au but (on rappelle que l'on ne cherche à instancier que les variables de type des théorèmes prouvés en amont) ;
  - elles ont le même nom ;
  - le type du destinataire est une instance du type de l'origine.
- Les arêtes de partage sont non orientées et relient deux constantes vérifiant les propriétés suivantes :
  - elles appartiennent à un même théorème ;
  - elles partagent une même variable de type.

Par convention, dans tous les exemples de graphes de dépendance donnés ci-dessous, nous représentons les constantes d'une même proposition sur une même ligne.

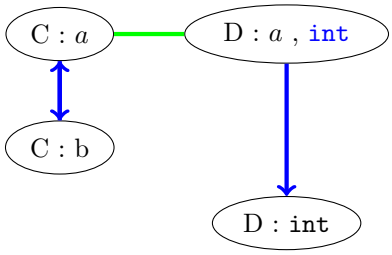


FIGURE 3 – Transformation de substitution associée à l'arête de substitution de droite

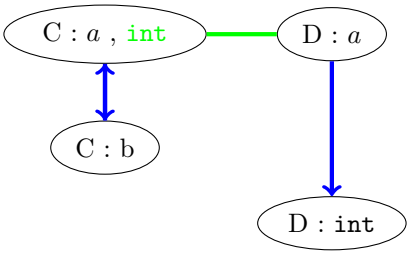


FIGURE 4 – Transformation de partage associée à l'arête de partage (au centre) et à l'arête de substitution de droite

L'idée de l'algorithme est d'appliquer une succession de transformations au graphe de dépendence, jusqu'à l'obtention d'un point fixe (s'il existe) nous donnant l'instanciation voulue. Nous définissons tout d'abord deux types de transformations sur un graphe de dépendence, une pour chaque type d'arête.

Définition. (Transformation de substitution) La transformation de substitution d'un graphe de dépendence G pour l'arête de substitution $\sigma : c \to c'$ instancie les types de c à l'aide des substitutions induites par σ. Ces types sont ensuite ajoutés au nœud c.

Définition. (Transformation de partage) La transformation de partage d'un graphe de dépendence G pour l'arête de partage $\rho : c \leftrightarrow c'$ et l'arête de substitution $\sigma : c' \to c''$ instancie les types de c à l'aide des substitutions induites par σ. Ces types sont ensuite ajoutés au nœud c.

Ces deux transformations sont appliquées simultanément afin de transformer un graphe de dépendance. Cette transformation est illustrée par l'exemple de la FIGURE 5.

Définition. (Transformation d'un graphe) La transformation $\mathcal{T}$ d'un graphe de dépendance G est définie comme l'application de toutes les transformations de substitution et de partage possibles à ce graphe de façon simultanée.

Définition. (Redondance) Dans un graphe, une arête de substitution est redondante si elle a la même origine et induit la même substitution qu'une arête de substitution interne au nœud de son origine. De telles arêtes peuvent être retirées du graphe pour obtenir un graphe équivalent.

La notion de redondance est illustrée par la FIGURE 6, dans laquelle nous avons ajouté des arêtes internes (en noir).

Les transformations successives d'un graphe de dépendance ne terminent pas toujours (ce qui rejoint le fait que la monomorphisation est incomplète). Nous donnons maintenant un exemple de

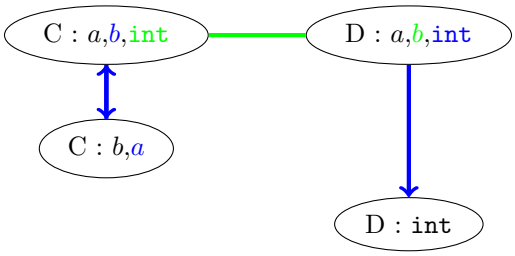


FIGURE 5 – Une itération de la transformation $\mathcal{T}$ sur le graphe de notre exemple

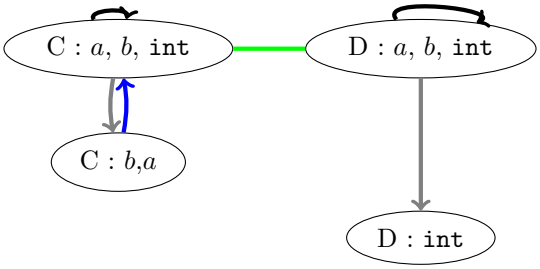


FIGURE 6 – Elimination des arêtes redondantes (en gris)

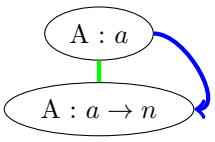
graphe de dépendance pour lequel les transformations successives font grossir indéfiniment les types possibles à chaque nœud.

Théorème. *(Circuit) Il existe un graphe de dépendance pour lequel la transformation $\mathcal{T}$ n'a pas de point fixe.*

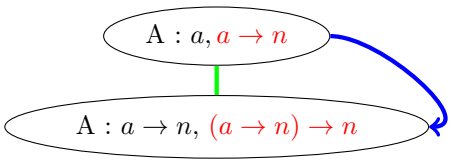
Démonstration. (Contre-exemple)

La transformation $\mathcal{T}$ n'a pas de point fixe pour le graphe suivant :

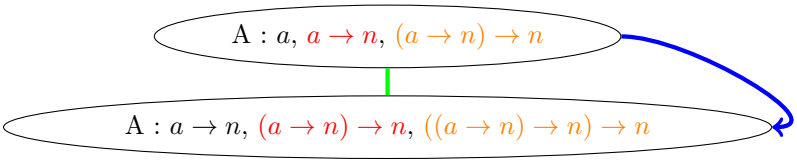
Graphe initial.



Etape 1.



Etape 2.



La substitution $\{a \mapsto (a \to n)\}$ peut être appliquée autant de fois que l'on veut à a sans trouver de point fixe. □

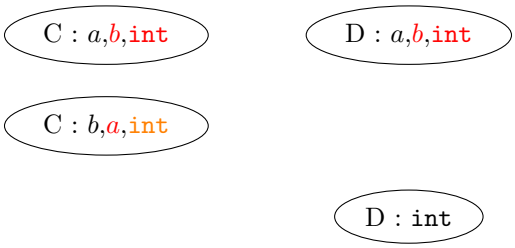


FIGURE 7 – Point fixe de la transformation $\mathcal{T}$ après 2 étapes de monomorphisation

$\mathcal{T}$ pouvant ne pas terminer, nous limitons le nombre d'itérations pour calculer la substitution finale. Cette borne est déterminée par la conjecture suivante.

Conjecture. *Soit p le nombre maximal d'arêtes de substitution d'un chemin sans circuit de G. Soit $G_p = \mathcal{T}^p(G)$. On conjecture que :*
- *Si G_p n'a pas de circuit interne à ses nœuds et n'a pas d'arête de substitution non redondante à l'extérieur de ses nœuds, alors G_p est un point fixe pour $\mathcal{T}$.*
- *Si G_p n'est pas un point fixe pour $\mathcal{T}$, alors pour tout $i > 0$, G_i n'est pas un point fixe pour $\mathcal{T}$.*

La FIGURE 7 montre le point fixe de $\mathcal{T}$ sur notre exemple.

Remarque. Le fait que l'on ait obtenu un point fixe après 2 étapes dans la FIGURE 7 vérifie la conjecture ci-dessus car le nombre maximal d'arêtes de substitution d'un chemin sans circuit du graphe initial (FIGURE 2) est de 2.

2.2.3. Création de nouveaux théorèmes à partir d'un graphe de dépendance

D'un noeud du graphe, nous pouvons extraire des arêtes de substitutions (internes ou externes) dont l'origine est ce noeud lesquels induisent un ensemble de substitutions pour ce noeud.

D'un ensemble de noeuds appartenant à un même théorème, on peut déduire un ensemble de substitutions en combinant les ensembles de subtitutions correspondants à chaque noeud.

Finalement nous instancions chaque théorème à l'aide de leur ensemble de substitutions correspondant et créons ainsi de nouveaux théorèmes susceptibles d'être utilisés par un prouveur ne supportant pas les types polymorphes.

2.2.4. Implantation

Nous présentons maintenant quelques détails pratiques pour implanter notre traduction.

Pour gérer les cas où la transformation sur le graphe de dépendance n'a pas de point fixe, nous avons limité à 15 la création de nouveaux théorèmes, ce qui limite du même coup le nombre d'itérations que le programme effectue avant de renvoyer le graphe de dépendance final.

Lors de la monomorphisation, un graphe de dépendance est implanté par une liste de listes, chaque liste contenant les constantes d'un même théorème ainsi que les constantes créées par la répétition de la transformation. Les arêtes ne sont pas représentées, car toutes les constantes d'un même théorème sont instanciées avec les substitutions générées par ces mêmes constantes, ce qui théoriquement revient à considérer que les constantes d'un même théorème sont liés 2 à 2 par une arête de partage.

Lors de l'impression, des dictionnaires injectifs, faisant correspondre les variables et les constantes de HOL4 avec des variables d'un fichier TFF, sont créés. Ceci augmente la sûreté de l'impression et permettra de rejouer ensuite la preuve (voir partie 4). De plus, les types doivent être traduits pour correspondre à la distinction au premier ordre entre variables, fonctions et prédicats.

3. Utilisation de Beagle et expériences

3.1. Présentation de Beagle

Beagle est un prouveur du premier ordre avec arithmétique très récent, prenant en entrée le format TFF. Des résultats expérimentaux (sur une version plus ancienne) ainsi que la théorie détaillée de Beagle peuvent être trouvés dans [2].

3.2. La tactique BEAGLE_TAC

Nous avons implanté une tactique BEAGLE_TAC exportant le but courant transformé par la traduction présentée dans la partie précédente au format TFF, puis appelant Beagle sur ce but transformé ; lorsqu'il répond que celui-ci est insatisfiable, il est ajouté comme axiome à HOL4 afin de résoudre le but initial (voir Figure 1).

Comme Beagle ne supporte pas l'arithmétique non linéaire, la constante multiplicative $*$ est traduite par une constante non interprétée lorsque le terme est non-linéaire. Cette modification devra être retirée si l'on souhaite utiliser un prouveur qui supporte l'arithmétique non linéaire.

3.3. Expériences

3.3.1. Software, hardware et tests

Les tests ont été effectués avec Beagle (version 0.7) et HOL4 (version de mai 2013 du dépot `https://github.com/mn200/HOL`), sur un processeur deux cœurs cadencé à 2.1 GHz avec 3.7 Go de mémoire vive. Nous avons imposé un timeout de 15 secondes par but à Beagle.

Lors de la construction de HOL4, certains buts sont résolus par la tactique METIS_TAC. La plupart de ces buts ne font intervenir que du raisonnement propositionnel, mais certains nécessitent de le combiner avec un raisonnement arithmétique, auquel cas les lemmes arithmétiques à utiliser sont fournis à METIS_TAC. Pour les expériences, nous avons utilisé BEAGLE_TAC sur 271 de ces buts, mais sans donner aucun lemme arithmétique. Afin de mesurer l'impact de notre algorithme de monomorphisation, nous avons lancé BEAGLE_TAC avec ou sans monomorphisation.

3.3.2. Résultats

Les résultats en termes de nombre de buts résolus sont présentés dans la Figure 8.

N'étant pas complet, Beagle peut parfois répondre "inconnu" s'il ne sait pas trouver la solution du problème. En revanche, les réponses "satisfiable" (dues à un manque d'information sur les constantes de HOL4 passées à Beagle) et "parsing error" (dues à un fichier problème trop grand ou à une erreur dans l'impression) correspondent à des erreurs dans notre utiliation de Beagle.

La Table 2 présente les résultats en termes de temps de calcul : on compare le temps mis par BEAGLE_TAC (décomposé en le temps de la traduction, celui de l'impression et celui mis par Beagle) au temps mis par METIS_TAC. La tactique METIS_TAC est plus rapide que BEAGLE_TAC. La traduction en elle-même prend plus de temps que METIS_TAC, mais c'est surtout Beagle qui apparaît comme le facteur de temps limitant. Comme nous le verrons plus bas, cela est principalement dû aux

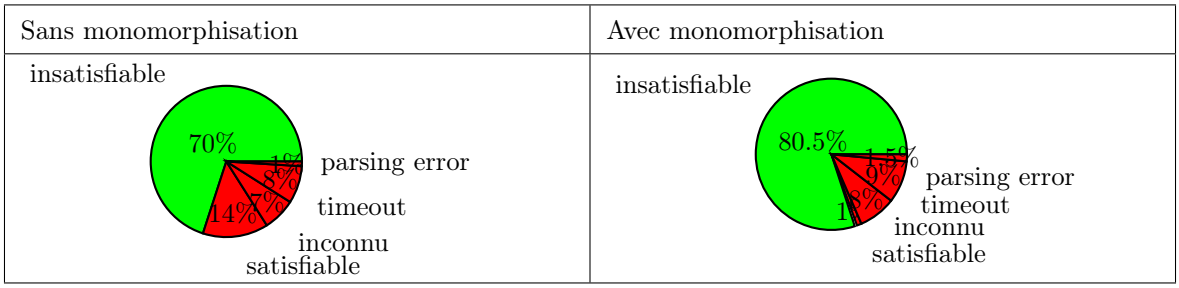


FIGURE 8 – Proportion de buts résolus par BEAGLE_TAC sans et avec monomorphisation

faiblesses de notre traduction actuelle. Il faut rappeler également que Beagle, contrairement à metis, doit effectuer lui-même le raisonnement arithmétique, mais cela n'a que peu d'influence comme montré ci-dessous.

BEAGLE_TAC	Traduction	Impression	Beagle	METIS_TAC
4,55	0,82	0,18	3,55	0,11

TABLE 2 – Temps moyens de résolution (en secondes)

Nous allons maintenant nous intéresser aux différents aspects mis en valeur par ces expériences.

Sur des problèmes d'ordre supérieur La TABLE 3 indique la proportion de problèmes résolus et les temps de calcul en séparant les problèmes qui sont initialement du premier ordre avec ceux d'ordre supérieur. Si BEAGLE_TAC a un comportement raisonnable sur les buts qui sont déjà du premier ordre, il n'arrive en revanche qu'à résoudre la moitié des buts contenant de l'ordre supérieur. Notre traduction est donc moins performante que celle de METIS_TAC (là encore, le résultat est légèrement biaisé par le fait que Beagle doit gérer l'arithmétique).

	Proportion	BEAGLE_TAC	Traduction	Impression	Beagle	METIS_TAC
Premier ordre	91,9%	2,711	0,148	0,08	2,483	0,13
O. supérieur	55,8%	11,07	3,2	0,51	7,36	0,04

TABLE 3 – Temps moyens de résolution de problèmes du premier ordre et d'ordre supérieur (en secondes)

Sur des problèmes polymorphes La TABLE 4 indique la proportion de problèmes résolus et les temps de calcul en séparant les problèmes qui sont initialement monomorphes avec ceux qui sont polymorphes. On constate qu'aussi bien la proportion de buts résolus que les temps de calcul sont voisins dans les deux cas, ce qui indique que notre algorithme de monomorphisation est efficace et bien adapté à notre traduction actuelle. Sa simplicité se reflète dans le temps mis par la traduction puisqu'il ne produit pas de délai supplémentaire.

	Proportion	BEAGLE_TAC	Traduction	Impression	Beagle	METIS_TAC
Monomorphe	81,1%	5,83	1,32	0,23	4,28	0,12
Polymorphe	79,9%	3,32	0,34	0,13	2,85	0,9

TABLE 4 – Temps moyens de résolution de problèmes monomorphes et polymorphes (en secondes)

La FIGURE 8 compare la proportion de buts résolus sans et avec monomorphisation. On remarque que l'ajout de l'étape de monomorphisation permet à BEAGLE_TAC de résoudre 10% de buts supplémentaires. On constate cependant que cette étape augmente la taille des buts, faisant ainsi croître le nombre de timeouts. Concernant l'algorithme en lui-même, un point fixe a été trouvé dans 102 cas des 139 problèmes ayant besoin d'être monomorphisés, soit 73% des cas.

Sur des problèmes arithmétiques La TABLE 5 indique la proportion de problèmes résolus et les temps de calcul en séparant les problèmes contenant de l'arithmétique de ceux qui n'en contiennent pas. Là encore, la proportion de buts résolus et les temps de calcul sont voisins dans les deux cas, ce qui montre l'efficacité de notre tactique à résoudre des buts arithmétiques.

	Proportion	BEAGLE_TAC	Traduction	Impression	Beagle	METIS_TAC
Non-arith.	81,6%	5,91	1,51	0,25	4,15	0,08
Arithmétique	79,6%	3,62	0,34	0,13	3,15	0,13

TABLE 5 – Temps moyens de résolution de problèmes non-arithmétiques et arithmétiques (en secondes)

Conclusion BEAGLE_TAC est à l'heure actuelle bien moins rapide que METIS_TAC. Outre les temps nécessaires à l'ouverture et la fermeture de fichiers, BEAGLE_TAC pèche sur les problèmes d'ordre supérieur, ce qui indique que notre traduction peut être encore grandement améliorée. Comme indiqué dans la partie 2, nous avons principalement mis l'accent sur la monomorphisation, ce qui se reflète dans les résultats.

En revanche, BEAGLE_TAC est plus expressive que METIS_TAC, et fournit la première tactique de HOL4 capable de combiner raisonnement au premier ordre et arithmétique.

Voici quelques directions possibles afin d'améliorer notre traduction :
- faire une élimination plus fine de l'ordre supérieur [6] ;
- inférer les lemmes utiles pour le problème à partir d'une heuristique sur les constantes [11] ;
- traduire les types rationnels et réels de HOL4 vers les types TFF correspondants (pour l'instant, seule l'arithmétique entière est gérée).

4. Vérification des preuves TFF

Jusqu'ici, le prouveur externe est utilisé par HOL4 comme un oracle : s'il résout le problème traduit, celui-ci est ajouté à HOL4 comme un axiome. Nous n'avons donc aucune garantie que son raisonnement, et donc l'axiome ajouté, sont corrects.

Pour utiliser des prouveurs externes sans compromettre la correction de HOL4, nous utilisons l'approche *sceptique* consistant à vérifier en HOL4 les traces éventuellement générées par ces prouveurs. Pour conserver la généricité, nous vérifions des traces fournies dans le format TFF.

Outre offrir plus de garanties à HOL4, la vérification de traces TFF peut être utilisés pour garantir les résultats donnés par des prouveurs du premier ordre lors d'autres utilisations.

4.1. Vérification des preuves

4.1.1. Parsing

Les traces supportées actuellement par notre tactiques sont les fichiers au format TFF contenant uniquement des clauses. Lorsque le parsing s'effectue à partir d'un problème généré par notre traduction, les variables, constantes et types du fichier de trace sont mises en correspondance avec celles du problème de départ à l'aide des dictionnaires injectifs HOL4 (voir partie 2.2.4).

4.1.2. Rejouer la preuve

Les tactiques METIS_TAC et COOPER_TAC (une implémentation de l'algorithme de Cooper pour l'arithmétique de Presburger [14]) sont utilisées conjointement pour rejouer les étapes de la preuve, en fonction de la nature de chacune des étapges.

Les deux tactiques utilisées ici ayant été prouvées correctes, cela permet d'importer des théorèmes prouvés par des prouveurs automatiques fournissant des traces au format TFF dans HOL4 sans en compromettre sa correction.

Cette utilisation de METIS_TAC à chaque petite étape propositionnelle de la preuve ne peut être efficace, mais permet de fournir un premier prototype permettant d'intégrer un prouveur externe de manière sceptique dans HOL4.

4.2. Application à Beagle

Nous appliquons ce procédé à la tactique BEAGLE_TAC, afin d'implanter complètement le schéma de la Figure 1. Cela nécessite tout d'abord de la part de Beagle d'être capable de générer des traces au format TFF, ce qui peut être fait de manière assez simple étant donné son fonctionnement. Nous expliquons de manière succincte comment procéder, avant de présenter notre prototype de tactique BEAGLE_TAC complète sur un exemple.

4.2.1. Génération de traces par Beagle

La boucle principale de Beagle maintient à jour deux ensembles de clauses, comme le montre le schéma de la Figure 9. À chaque itération, Beagle sélectionne une clause dans le nouvel ensemble de clauses, l'ajoute à l'ancien ensemble de clauses, puis en déduit de nouvelles clauses qui sont ajoutées au nouvel ensemble. Les clauses exportées dans nos traces sont donc celles de l'ancien ensemble, afin de ne garder que les étapes de la preuve qui ont été utilisées [4].

4.2.2. Description sur un exemple

Nous allons décrire comment rejouer la trace du problème suivant :

$$((x = 2) \vee (x = 4)) \wedge y = 2 * x) \Rightarrow (y = 4) \wedge (y = 8)$$

donnée en Figure 10.

La trace est rejouée étape par étape :

– Tout d'abord les axiomes 1 à 7 sont lus et prouvés à partir des axiomes de HOL4 desquels ils ont été traduits.

4. Il est parfois nécessaires d'ajouter quelques étapes manquantes, comme la déduction de la clause vide lors d'un raisonnement par l'absurde.

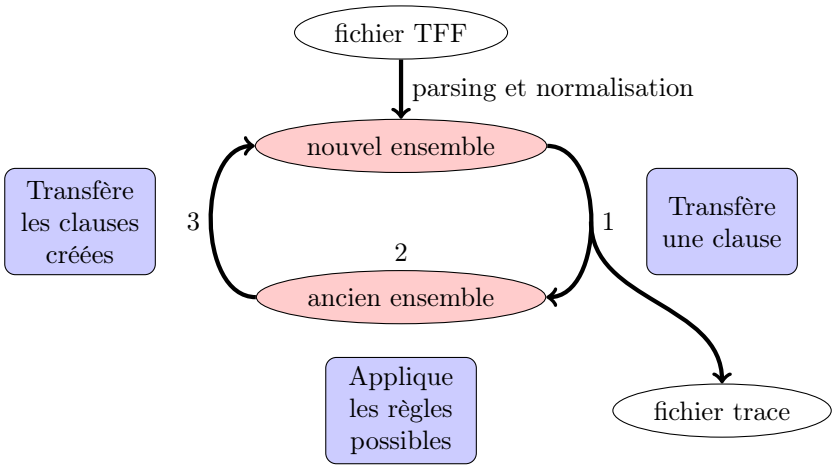


FIGURE 9 – Boucle principale de Beagle

Fichier preuve	Arbre de la preuve

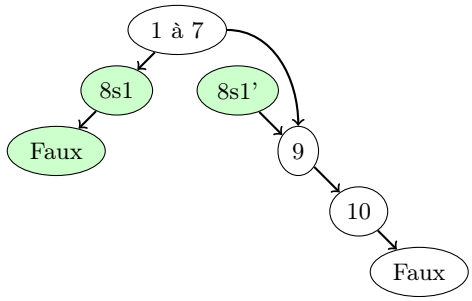
`tff(1,axiom, ...). ...`
`tff(4,axiom,(xy!=8),...).`
`tff(5,axiom,(xy!=4),...).`
`tff(6,axiom,(xx=2 | xx=4),...).`
`tff(7,axiom,(product(2, xx)=xy),...).`
`tff(8s1,plan,((xx=2),Leftsplit).`
`tff(9,plain,((xx=4),...).`
`tff(10,plain,((xx!=2),...).`
	

FIGURE 10 – Comment rejouer une preuve

- A l'étape $8s1$, l'indication `Leftsplit` nous signale que ceci est le début d'un raisonnement par l'absurde. Nous ajoutons donc cette hypothèse à une pile sans la prouver.
- Avant de passer à l'étape suivante, la numérotation nous indique que le raisonnement par l'absurde a pris fin. Nous devons donc prouver que l'on a aboutit à une contradiction, modifier l'hypothèse par l'absurde du haut de la pile en son contraire et l'ajouter comme proposition déduite de nos axiomes $8s1'$.
- Les étapes 9 et 10 sont prouvées en utilisant toutes les étapes parentes dans l'arbre de la FIGURE 10.
- Pour conclure, il suffit de montrer que la conjonction des propositions de la branche droite permet de déduire une contradiction.

4.2.3. Conclusion

Nous avons décrit un moyen simple utilisant les prouveurs internes de HOL4 pour rejouer les traces générées par Beagle, après légère transformation de ce dernier. Son efficacité pourra être largement améliorée, notamment en s'inspirant de [1, 11]

5. Travaux similaires et futurs

L'interaction sceptique entre assistants de preuve et prouveurs automatiques externes a déjà été largement étudiée, pour différents types d'assistants de preuve – basés sur la théorie des types [1, 3] ou sur la logique d'ordre supérieur [10, 11] – et différents types de prouveurs automatiques – du premier ordre [11], de Satisfiabilité Modulo Théories [1,6,11], de terminaison [10], d'arithmétique [3]. . .. L'interaction présentée dans cet article est destinée en particulier à être utilisée avec des prouveurs du premier ordre avec arithmétique.

Une traduction prouvée de l'ordre supérieur vers le premier ordre est déjà implantée pour la plupart des applications citées ci-dessus. Notre traduction reprend les idées présentée dans ces travaux (λ-lifting, traitement de l'arithmétique non linéaire [6], instantiation des arguments booléens), mais en réalisant un traitement particulier pour l'arithmétique linéaire comme expliqué ci-dessus, ainsi qu'une monomorphisation décrite de manière simple à l'aide de points fixes (partie 2.2). Si les expériences présentées dans cet article montrent l'efficacité absolue de notre algorithme de monorphisation, nous nous réservons comme travaux futurs une comparaison avec les algorithmes de monomorphisation existants. Un point particulièrement intéressant serait une comparaison avec une traduction vers le format TFF1 proposé récemment par Blanchette et Paskevich [4]. Les autres étapes de la traduction doivent également être améliorées.

La vérification de traces présentée ici constitue un premier prototype basé sur l'idée de rejouer la preuve à l'aide des prouveurs automatiques internes à HOL4, comme c'est le cas pour la tactique sledgehammer [11]. Bien que beaucoup moins efficace pour l'instant, elle illustre l'aspect selon lequel notre tactique permet de combiner raisonnement propositionnel et arithmétique : la vérification de traces peut se faire à l'aide des tactiques METIS_TAC et COOPER_TAC, mais sans nécessité d'une tactique interne combinant les deux.

6. Conclusion

Cet article présente une traduction de la logique d'ordre supérieure polymorphe vers la logique du premier ordre monomorphe avec arithmétique, faisant notamment intervenir un algorithme de monomorphisation simple mais néanmoins efficace. Cette traduction est implantée et prouvée correcte en HOL4, permettant à cet assistant de preuve de bénéficier de prouveurs du premier ordre avec (ou sans) arithmétique tout en ayant confiance dans la traduction. L'application à Beagle permet de fournir une tactique utilisable et efficace à HOL4, combinant raisonnement propositionnel et arithmétique linéaire. Un premier prototype permet de vérifier les traces au format TFF afin de garantir la correction, prototype également appliqué à Beagle.

Le travail présenté ici constitue donc une base pour l'interaction entre HOL4 et des prouveurs externes, qui peut être encore largement développée. Les résultats expérimentaux sont encourageants quant aux perspectives offertes par une telle intégration. Ce travail constitue également une première utilisation réelle de Beagle, montrant son efficacité et son utilité.

Remerciements Les auteurs sont particulièrement reconnaissants envers Peter Baumgartner et Josh Bax pour leurs explications sur le fonctionnement de Beagle, ainsi que leur aide pour la résolution de certaines des erreurs produites par des versions préliminaires de la traduction. Les auteurs remercient également Alexis Saurin pour ses suggestions à propos de cet article.

Références

[1] M. Armand, G. Faure, B. Grégoire, C. Keller, L. Théry, and B. Werner. A modular integration of sat/smt solvers to coq through proof witnesses. In J.-P. Jouannaud and Z. Shao, editors, *CPP*, volume 7086 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 2011.

[2] P. Baumgartner and U. Waldmann. Hierarchic superposition with weak abstraction. In Bonacina [7], pages 39–57.

[3] F. Besson. Fast reflexive arithmetic tactics the linear case and beyond. In T. Altenkirch and C. McBride, editors, *TYPES*, volume 4502 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2006.

[4] J. C. Blanchette and A. Paskevich. Tff1 : The tptp typed first-order form with rank-1 polymorphism. In Bonacina [7], pages 414–420.

[5] F. Bobot and A. Paskevich. Expressing polymorphic types in a many-sorted language. In C. Tinelli and V. Sofronie-Stokkermans, editors, *FroCoS*, volume 6989 of *Lecture Notes in Computer Science*, pages 87–102. Springer, 2011.

[6] S. Böhme. *Proving Theorems of Higher-Order Logic with SMT Solvers*. PhD thesis, Technische Universität München, 2012.

[7] M. P. Bonacina, editor. *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *Lecture Notes in Computer Science*. Springer, 2013.

[8] J. Hurd. First-order proof tactics in higher-order logic theorem provers. *Design and Application of Strategies/Tactics in Higher Order Logics, number NASA/CP-2003-212448 in NASA Technical Reports*, pages 56–68, 2003.

[9] J. Hurd. System Description : The Metis Proof Tactic. *Empirically Successful Automated Reasoning in Higher-Order Logic (ESHOL)*, pages 103–104, 2005.

[10] C. Kaliszyk and J. Urban. Learning-assisted automated reasoning with flyspeck. *CoRR*, abs/1211.7012, 2012.

[11] J. C. B. Lawrence C. Paulson. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In S. S. G. Sutcliffe, E. Ternovska, editor, *IWIL-2010*, 2010.

[12] S. Lescuyer. *Formalizing and Implementing a Reflexive Tactic for Automated Deduction in Coq*. PhD thesis, Université Paris-Sud, 2011.

[13] J. Meng and L. C. Paulson. Translating higher-order clauses to first-order clauses. *J. Autom. Reasoning*, 40(1) :35–60, 2008.

[14] M. Norrish. Complete integer decision procedures as derived rules in hol. In *Theorem Proving in Higher Order Logics, TPHOLs 2003, volume 2758 of Lect. Notes in Comp. Sci*, pages 71–86. Springer-Verlag, 2003.