Reconstruction de preuves pour les formules quantifiées et ensemblistes

Clément Hurlin

Mémoire

pour l'obtention du

Master de l'Université Nancy I
Henri Poincaré

discipline

Informatique
spécialité « maîtrise du logiciel »

par
Clément Hurlin
le 29 Juin 2006

Reconstruction de preuves pour les formules quantifiées et ensemblistes

Encadrants :
Loria (France) :
Pascal Fontaine
Stephan Merz
TUM (Allemagne) :
Tjark Weber
Amine Chaieb

Jury :
Dominique Mery
Noëlle Carbonell
Guy Perrier
Didier Galmiche
Claude Godard
Stephan Merz

Stage effectué au Loria et à l'Université Technique de Munich.
Année 2005/2006

Table des matières

Je remercie mes encadrants, en particulier Pascal Fontaine pour les intéressantes discussions que nous avons eues, Amine Chaieb pour ses conseils qui tombent toujours à pic et Tjark Weber pour ses indications techniques très utiles. Je tiens également à remercier Le Transi[1] et ma *principessa*...

[1] Il a su rester de marbre.

1 Introduction

Les prouveurs interactifs tels que PVS [ORS92], CoQ [The05] ou Isabelle [NPW02] ont un champ d'application très large. Ils n'imposent pas l'utilisation d'une logique (c'est à dire un cadre mathématique fixé) et le langage d'entrée est très expressif. On peut les utiliser pour représenter de manière intuitive une grande partie des mathématiques. Cependant ces prouveurs ne sont pas automatiques : mis à part quelques stratégies, ils demandent une interaction de l'utilisateur. Ainsi, faire une preuve non triviale demande un certain degré d'expertise.

Pour utiliser un prouveur (par exemple Isabelle), il faut déclarer un but et des hypothèses puis appliquer des règles pour montrer que les hypothèses données impliquent le but. C'est pourquoi on parle d'« assistant » de preuves. Isabelle n'a pas de méthode automatique efficace pour décharger des obligations de preuves de grande taille, ce qui veut dire que l'utilisateur doit effectuer chaque opération (même très simple) de la preuve lui-même. Faire une preuve en Isabelle consiste à essayer d'appliquer des règles et Isabelle assure que chaque application d'une règle, si elle réussit, est correcte, mais ne guide pas du tout le processus. C'est un répétition de séquences « essai - échec » fastidieux car il faut bien connaître comment fonctionne le système pour traduire un raisonnement mathématique simple en une suite de règles Isabelle. C'est pourquoi augmenter l'automatisation d'Isabelle est un point crucial pour pouvoir utiliser des méthodes formelles à grande échelle (vérification de programmes, de théorèmes etc). En effet, l'utilisateur d'Isabelle est souvent un novice en la matière, et son but n'est pas de devenir un expert du système, mais simplement de décharger son obligation de preuve.

Il existe également des prouveurs automatiques très efficaces. Depuis quelques années les performances des prouveurs pour la logique propositionnelle (SAT) se sont considérablement améliorées et les prouveurs pour la logique du premier ordre (avec des restrictions) en ont profité. Parmi ceux-ci, on trouve les prouveurs SMT -*Satisfiability modulo theories*- [RT05]. Ces prouveurs déterminent la satisfaisabilité ou l'insatisfaisabilité d'une formule par rapport à une théorie (théorie des tableaux ou des listes par exemple). L'interaction entre l'utilisateur et un prouveur SMT est réduite au strict minimum. Elle consiste seulement à lancer celui-ci sur un fichier texte décrivant l'obligation de preuves. Une fois la recherche de preuve effectuée, on sait juste si la formule est satisfaisable ou insatisfaisable. Dans le cas où elle est insatisfaisable (c'est ce qui nous intéresse dans ce document), nous n'avons aucune indication sur les étapes réalisées. En fait, le fonctionnement et l'utilisation d'un prouveur automatique sont aux antipodes de l'utilisation d'un prouveur interactif tel que Isabelle. En effet, on ne sait pas quelles sont les étapes difficiles de la preuve, on sait juste le résultat.

L'intégration des prouveurs automatiques au sein des prouveurs interactifs permet d'allier les avantages des deux approches : d'un côté on conserve la richesse du langage d'entrée des outils interactifs et de l'autre on bénéficie de l'automatisation des prouveurs automatiques. Cette technique a déjà été mise en œuvre (voir [Hur99, NBH00]) mais par rapport aux travaux cités, nous utilisons une technique différente pour trouver les preuves et le fragment que l'on traite n'avait pas été étudié.

Dans l'assistant générique de preuves Isabelle, plusieurs théories sont prédéfinies : la théorie des ensembles ZF, la logique du premier ordre FOL et la logique

d'ordre supérieur HOL. Dans la suite de ce document, je parlerai uniquement de l'instanciation d'Isabelle pour la logique d'ordre supérieur : Isabelle/HOL. Isabelle est un prouveur de style *LCF*. C'est-à-dire que la correction du système est assuré par le fait que les preuves de tous les théorèmes ne font intervenir qu'un nombre restreint de règles. De cette manière, pour vérifier la correction d'Isabelle/HOL, il « suffit » de vérifier la correction de ces règles. En plus de ces règles, une bibliothèque a été implantée pour pouvoir les manipuler facilement. Cette interface permet de rajouter, moyennant une bonne connaissance du système, de nouvelles stratégies de recherche de preuves.

L'objectif du travail est de déléguer les obligations de preuves d'Isabelle/HOL à un prouveur automatique (comme réalisé dans [MQPss, Hur99]) : haRVey. Cependant, on souhaite garder le haut niveau de confiance que Isabelle/HOL donne. On pourrait prendre le résultat d'haRVey tel quel mais pour assurer la correction de la preuve effectuée, on va faire de la « reconstruction » de preuves. C'est-à-dire que Isabelle/HOL va déléguer la recherche de la preuve à haRVey mais une fois celle-ci trouvée, nous allons la rejouer dans Isabelle/HOL. Ainsi la preuve est entièrement réalisée au niveau d'Isabelle/HOL et tout est certifié par le noyau de ce dernier. De plus, au lieu de devoir essayer plusieurs règles à la suite, il suffira à l'utilisateur d'appliquer notre règle qui sera entièrement automatique. Cette technique permet de prouver des formules d'une taille surprenante pour un prouveur interactif.

Ainsi, on fournit à l'utilisateur d'Isabelle de nombreuses facilités. En effet, les tactiques automatiques d'Isabelle ne sont pas prévues pour décharger des obligations de preuves très larges. Intégrer haRVey à Isabelle/HOL permet à l'utilisateur de raccourcir considérablement le temps nécessaire pour effectuer une preuve en Isabelle/HOL.

Une interface existe déjà entre Isabelle/HOL et haRVey. Cette interface permet d'envoyer à haRVey des formules contenant des symboles de fonctions et de prédicats non interprétés ainsi que des égalités. Comme cela n'est pas un fragment très expressif, nous avons étendu cette interface pour gérer des formules plus riches. Cela a été réalisé de manière graduelle : la reconstruction a été mise en œuvre pour des fragments de la logique du premier ordre puis des formules ensemblistes. Pour cela, il a fallu modifier haRVey et à écrire de nouvelles stratégies en Isabelle/HOL. La structure du document suit cette progression de l'expressivité des formules.

Dans la section 2, je décris les méthodes pour résoudre les formules QF-FOL (formules contenant uniquement des symboles de fonctions et de prédicats non interprétés avec égalité). Au cours de la section 3, j'explique l'équivalence entre une formule SET (contenant des symboles ensemblistes) et une formule FOL (formules de la logique du premier ordre avec égalité). La section 4 est consacrée à la mise en pratique de ces techniques. Je présente successivement la reconstruction pour des formules ∀FOL (formules FOL sans quantificateur existentiel), FOL puis SET. Je décris comment les techniques décrites précedemments ont été implanté.

2 Méthode de recherche de preuves

Le but d'Isabelle/HOL est de démontrer des théorèmes, ce qui revient à montrer la validité d'une formule. haRVey détermine la satisfaisabilité de la

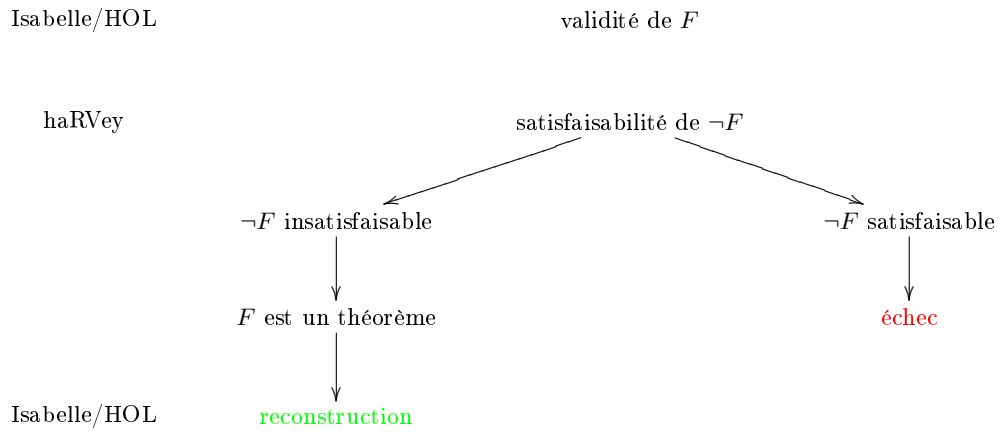

FIG. 1 – Procédure générale

formule qu'on lui passe en entrée. Or on sait qu'une formule est valide si et seulement si sa négation est insatisfaisable.
Soit F une formule donnée en entrée à Isabelle. Pour montrer le théorème $\vdash F$, on va montrer le théorème $\neg F \vdash \bot$. La première étape consiste à envoyer $\neg\ F$ à haRVey.

Si haRVey renvoie que la formule est satisfaisable, F n'est pas valide, la reconstruction n'a pas lieu. Si haRVey renvoie que la formule est insatisfaisable, il génère une trace de la recherche de preuve effectuée (cf. figure 2 page 5). Cette recherche met en œuvre un algorithme de clôture de congruence détaillé dans la sous section suivante. La trace de la preuve trouvée consiste en plusieurs formules $C_1 \dots C_n$. Ces formules sont les clauses de conflit. haRVey donne les étapes à effectuer pour prouver chaque C_i.

Chaque théorème $\vdash C_i$ est établi dans Isabelle/HOL en utilisant la trace donnée par haRVey. On obtient les théorèmes suivants : $\vdash C_1, \dots, \vdash C_n$. On peut alors construire le théorème : $\neg F, C_1, \dots, C_n \vdash \bot$. Cette étape ne requiert que du raisonnement propositionnel comme décrit au paragraphe 2.2. Des preuves des théorèmes faites en 3 et du théorème de l'étape précédente, on déduit le théorème $\vdash F$.

2.1 Algorithme de clôture de congruence

Cet algorithme permet de construire les classes d'équivalences d'un ensemble de termes (contenant uniquement le symbole d'égalité et des symboles de fonctions et de prédicats non interprétés) représentant les égalités entre ces termes. Deux termes appartiennent à la même classe d'équivalence si on peut déduire leur égalité de l'ensemble d'hypothèses initial.

Notre but est de dériver l'insatisfaisabilité d'un ensemble de littéraux. Un ensemble de littéraux peut être insatisfaisable pour deux raisons. Premièrement si il contient deux littéraux complémentaires du même prédicat et dont les arguments appartiennent à la même classe d'équivalence. C'est pourquoi $\{a = b, p(a), \neg p(b)\}$ n'est pas satisfaisable. Deuxièmement s'il y a une dis-égalité entre deux termes appartenant à la même classe comme dans $\{a = b, f(a) \neq f(b)\}$.

Sans rentrer dans les détails, l'algorithme permet de construire au fur et à mesure les classes d'équivalences correspondant aux égalité entre termes. Initialement tous les termes sont dans une classe d'équivalence différente. Ensuite les partitions sont mises à jour en prenant en compte les égalités entre les termes et la congruence : $a = b \implies f(a) = f(b)$. Comme expliqué dans [FMM+06], on peut décomposer le raisonnement utilisé dans l'algorithme de clôture de congruence en trois règles : congruence **C**, transitivité et symétrie.

Exemple :
Considérons l'ensemble de termes suivant : $\{a, b, c, f(a), f(b), f(c)\}$ et l'ensemble d'égalités suivant : $\{a = b, f(b) = f(c)\}$. On relie deux classes d'équivalence par un arc si elles sont égales. Initialement chaque terme a une classe d'équivalence propre :

$$a \qquad b \qquad c$$

$$f(a) \qquad f(b) \qquad f(c)$$

Puis si on prend en compte que $a = b$ et $f(b) = f(c)$, on a :

$$a \text{ ——— } b \qquad c$$

$$f(a) \qquad f(b) \text{ —— } f(c)$$

Comme $a = b$, la règle de congruence permet de déduire $f(a) = f(b)$:

$$a \text{ ——— } b \qquad c$$

$$f(a) \overset{\mathbf{C}}{\text{——}} f(b) \text{ —— } f(c)$$

On s'aperçoit désormais que le graphe permet de décomposer le raisonnement. Ainsi, étant donné les hypothèses $a = b$ et $f(b) = f(c)$, si on veut obtenir l'égalité entre $f(c)$ et $f(a)$ il faut suivre le chemin entre les nœuds correspondants. Un arc simple sous entend qu'il a été obtenu à partir d'une hypothèse, un arc marqué d'un **C** indique qu'il fait intervenir la congruence, le fait de passer d'un arc à un autre fait intervenir la transitivité de l'égalité et la congruence si l'arc est marqué d'un **C**.

Il a été vérifié que cette méthode admet les propriétés suivantes (étant donné un ensemble d'hypothèses $\mathcal{H}$) :

- $\mathcal{H}$ implique une égalité si et seulement si il existe un chemin entre les deux nœuds correspondants dans le graphe.
- Il y a au plus un chemin entre deux nœuds du graphe.
- L'égalité entre deux termes est la conséquence des règles de transitivité, symétrie et réflexivité appliquées aux nœuds constituant le chemin entre ces deux termes et à la congruence.

Comme haRVey utilise un algorithme de clôture de congruence, la trace de la preuve trouvée contient les informations du graphe final, c'est à dire l'application des différentes règles (transitivité, symétrie, congruence et réflexivité) à un certain nombres de termes. L'étape trois de la reconstruction de preuve consiste donc à rejouer dans Isabelle/HOL chaque étape.

On peut voir un résumé de la procédure de reconstruction[2] sur la figure 2 (où un □ indique ou la preuve est terminée).

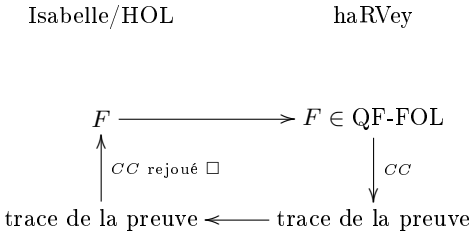

FIG. 2 – État initial de la reconstruction

2.2 Abstraction booléenne

Ce paragraphe explique comment une procédure de décision pour la logique propositionnelle peut être utilisée sur des formules du premier ordre sans quantificateurs. Imaginons que l'on souhaite vérifier la satisfaisabilité de la formule suivante :

$$x = y \wedge [f(x) \neq f(y) \vee (\neg p(x) \wedge p(z))] \tag{1}$$

Une abstraction booléenne de la formule peut être construite en remplaçant les atomes par une variable booléenne. On obtient :

$$p_1 \wedge [\neg p_2 \vee (\neg p_3 \wedge p_4)] \tag{2}$$

où les variables p_1, p_2, p_3 et p_4 représentent respectivement les atomes $x = y$, $f(x) = f(y)$, $p(x)$ et $p(z)$. Cette abstraction booléenne a deux modèles : $\{p_1, \neg p_2\}$ et $\{p_1, \neg p_3, p_4\}$. Le premier modèle, qui rend vrai p_1 et rend faux p_2, ne correspond pas à un modèle de la formule initiale (1) car il implique que $x = y$ soit vrai et que $f(x) = f(y)$ soit faux. Autrement dit le modèle abstrait n'est pas un modèle concret de (1). Par contre, au second modèle abstrait correspond un modèle de (1) car $\{x = y, \neg p(x), p(z)\}$ est satisfaisable. De manière générale, une formule est satisfaisable s'il existe un modèle de l'abstraction booléenne qui soit un modèle de la formule initiale. C'est pourquoi la formule (1) est satisfaisable.

A chaque clause de conflit correspond un modèle abstrait qui n'est pas un modèle de la formule initiale. De plus, on sait que pour transformer le problème de la satisfaisabilité d'une formule du premier ordre sans quantificateur en un problème purement propositionnel, il suffit de prendre en compte la conjonction de tous ses modèles abstraits.

Soit F une formule. Soient $A_1, \ldots, A_n$ tous ses modèles abstraits. Alors la

[2] où QF-FOL désigne les formules de la logique du premier ordre sans quantificateurs et où CC désigne l'algorithme de clôture de congruence.

validité de la formule $F' = A_1 \wedge \cdots \wedge A_n$ peut être décidée de manière purement propositionnelle. De plus, F est satisfaisable si et seulement si F' est satisfaisable. C'est ce qui nous permet d'affirmer que la procédure décrite au paragraphe 2 est correct.

2.3 Reconstruction d'une clause de conflit

Ce paragraphe décrit le format de trace de preuve de haRVey et la reconstruction correspondante dans Isabelle/HOL (comme écrit dans [FMM+06]). Il est directement hérité du raisonnement utilisé (l'algorithme de clôture de congruence). Pour chaque clause de conflit haRVey génère une trace des étapes de la preuve de cette forme :

```
TRANS: <séquent>
CONGR: <séquent>
PRED : <séquent>
INEQ : <séquent>
```

et qui termine avec :

```
CONFL: <formule>
```

La formule suivant le mot clé `CONFL` est la clause de conflit. Avant d'expliquer la signification de chaque mot clé, regardons la structure générale de la trace de la preuve. La preuve est faite à l'envers, comme en calcul des séquents quand on remonte vers les axiomes. Par exemple si la liste des séquents précédant la clause de conflit est :

$$\begin{array}{rc} l_1 : & C_{11}, \ldots, C_{1k_1} \vdash B_1 \\ \vdots & \vdots \\ l_n : & C_{n1}, \ldots, C_{nk_n} \vdash B_n \end{array}$$

(où chaque label l_i est soit `TRANS`, `CONGR`, `PRED` ou `INEQ`), le premier séquent énonce toujours une contradiction (B_1 est $\bot$). Les hypothèses du séquent i vérifient toujours la propriété suivante : leurs négations apparaissent dans la clause de conflit ou dans la conclusion d'un séquent précédent : quelle que soit l'hypothèse C_{ik} du séquent i il existe B_j, avec $j > i$ tel que $C_{ik} = B_j$. La clause de conflit est donc prouvée par contradiction, car la preuve consiste à partir de sa négation jusqu'à dériver $\bot$.

Regardons maintenant comment chaque séquent est démontré. Le mot clé précédent chaque séquent indique le raisonnement à effectuer.

1. Le mot clé `PRED` indique que le séquent est prouvable par substitution puis par contradiction, par exemple :

$$s = t, Ps, \neg Pt \vdash \bot$$

2. Le mot clé `INEQ` indique que le séquent contient une paire d'égalités contradictoires :

$$s = t, s \neq t \vdash \bot$$

3. Le mot clé `TRANS` indique que le séquent est démontrable grâce à la réflexivité, la symétrie et la transitivité de l'égalité.

4. Le mot clé `CONGR` indique que le séquent est prouvable en utilisant la règle de congruence. Comme les termes sont représentés de manière curryfiés, on a besoin d'une seule règle, indépendemment de l'arité des symboles considérés :

$$x = y \vdash f\ x = f\ y$$

Exemple : En lançant haRVey sur la formule suivante :

$$a = f\ a \wedge f\ a = f\ (g\ b) \wedge f\ (g\ b) = g\ (f\ a) \wedge g\ b = g\ (g\ a) \vdash a = g\ b,$$

on obtient une clause de conflit qui est la formule elle-même mais en forme CNF. La trace de la recherche de preuve est visible sur la figure 3. On peut remarquer que les négations des égalités et dis-égalités en gras apparaissent toutes dans la clause de conflit. Les égalités et dis-égalités restantes apparaissent dans les conclusions des séquents précédents.

`INEQ :`	$a = g\ b,\ \mathbf{a \neq g\ b} \vdash \bot$
`TRANS :`	$\mathbf{a = f\ a},\ \mathbf{f\ a = f\ (g\ b)},\ \mathbf{f\ (g\ b) = g\ (f\ a)},$
	$g\ (f\ a) = g\ (g\ a),\ \mathbf{g\ b = g\ (g\ a)} \vdash a = g\ b$
`CONGR :`	$f\ a = g\ a \vdash g\ (f\ a) = g\ (g\ a)$
`TRANS :`	$g\ (f\ a) = g\ a,\ \mathbf{f\ (g\ b) = g\ (f\ a)},\ \mathbf{f\ a = f\ (g\ b)} \vdash f\ a = g\ a$
`CONGR :`	$\mathbf{a = f\ a} \vdash g\ (f\ a) = g\ a$
`CONFL :`	$a = g\ b \vee a \neq f\ a \vee f\ a \neq f\ (g\ b) \vee f\ (g\ b) \neq g\ (f\ a) \vee g\ b \neq g\ (g\ a)$

FIG. 3 – Trace de la recherche de preuve pour une clause de conflit

3 Transformation de formules

L'objectif de la manipulation est de résoudre des formules ensemblistes, c'est à dire des formules contenant des opérateurs sur les ensembles. Comme haRVey sait résoudre les formules de la logique du premier ordre, il faut transformer les formules ensemblistes en formules du premier ordre. Pour cela on décrit les ensembles par leur fonction caractéristique : un ensemble A est considéré comme un prédicat A_p tel que $a \in A \Leftrightarrow A_p(a)$ (afin que la distinction soit claire, on associe au symbole d'ensemble E, le symbole de prédicat $\breve{E}$ dans la suite). En augmentant cette règle pour d'autres opérateurs ensemblistes, on peut transformer des formules ensemblistes en formule du premier ordre. Par exemple, $a \in A \setminus \{b\} \Leftrightarrow \breve{A}(a) \wedge a \neq b$. Cette transformation se déroule ainsi :

Formule ensembliste (SET)
↓
Désucrage *(apparition de λ-termes)*
↓
β-réduction
↓
Application de l'extensionnalité
↓
Formule du premier ordre (FOL).

Le principe d'extensionnalité permet de réduire des égalités du second ordre à des égalités du premier ordre. Par exemple si P et Q sont des prédicats, $P = Q \Leftrightarrow \forall x.\big[P(x) = Q(x)\big]$.

3.1 Des lambdas termes

L'étape de désucrage consiste à remplacer syntaxiquement les symboles ensemblistes par des λ-termes. Cela revient à faire de la réécriture. Au cours de celle-ci on transforme des symboles d'ensembles en symboles de prédicats. L'ensemble des règles de réécriture est donné dans la définition suivante :

Définition 3.1 (réécriture des opérateurs ensemblistes)**.**

Appartenance	$a \in E$	$\longrightarrow$	$\big(\lambda xP.P\ x\big)(a, E)$
Sous ensemble	$E \subseteq F$	$\longrightarrow$	$\big(\lambda PQ.\, \forall x.\, P(x) \Rightarrow Q(x)\big)(E, F)$
Sous ensemble strict	$E \subset F$	$\longrightarrow$	$\big(\lambda PQ.\, (\forall x.\, P(x) \Rightarrow Q(x)) \wedge (\exists x.\, \neg P(x) \wedge Q(x))\big)(E, F)$
Egalité	$E = F$	$\longrightarrow$	$\big(\lambda PQ.\, \forall x.\, P(x) = Q(x)\big)(E, F)$
Intersection	$E \cap F$	$\longrightarrow$	$\big(\lambda PQ.\, \lambda x.\, P(x) \wedge Q(x)\big)(E, F)$
Union	$E \cup F$	$\longrightarrow$	$\big(\lambda PQ.\, \lambda x.\, P(x) \vee Q(x)\big)(E, F)$
Différence	$E \setminus F$	$\longrightarrow$	$\big(\lambda PQ.\, \lambda x.\, P(x) \wedge \neg Q(x)\big)(E, F)$
Explicitation	$\{a_1, \ldots, a_n\}$	$\longrightarrow$	$\lambda x.\, (x = a_1 \vee \cdots \vee x = a_n)$
$\emptyset$	$\emptyset$	$\longrightarrow$	$\lambda x.\, \bot$
Ω	Ω	$\longrightarrow$	$\lambda x.\, \top$

Exemple :
A chaque étape les sous-formules transformées sont soulignées et la règle appliquée est explicitée à droite de la formule. La lettre D indique une étape de réécriture, un β indique une β-réduction, un E l'application de l'extensionnalité :

$$
\begin{array}{ll}
A \setminus \{a, b\} = B \cap C & \text{D D} \\
A \setminus \underline{\lambda x.(x = a \vee x = b)} = \underline{\big(\lambda PQ.\lambda y.P(y) \wedge Q(y)\big)(B,\, C)} & \text{D } \beta \\
\underline{\big(\lambda PQ.\lambda z.P(z) \wedge \neg Q(z)\big)\big(A, \lambda x.(x = a \vee x = b)\big)} = \lambda y.\breve{B}(y) \wedge \breve{C}(y) & \beta \\
\lambda z.\breve{A}(z) \wedge \neg \underline{\lambda x.(x = a \vee x = b)(z)} = \lambda y.\breve{B}(y) \wedge \breve{C}(y) & \beta \\
\underline{\lambda z.\breve{A}(z) \wedge (z \neq a \wedge z \neq b)} = \underline{\lambda y.\breve{B}(x) \wedge \breve{C}(y)} & \text{E} \\
\forall w.\big[\underline{\big(\lambda x.\breve{A}(x) \wedge (x \neq a \wedge x \neq b)\big)(w)} = \underline{\big(\lambda y.\breve{B}(y) \wedge \breve{C}(y)\big)(w)}\big] & \beta\ \ \beta \\
\forall w.\big[\big(\breve{A}(w) \wedge w \neq a \wedge w \neq b\big) = \big(\breve{B}(w) \wedge \breve{C}(w)\big)\big] &
\end{array}
$$

On peut noter que l'utilisation de l'axiome d'extensionnalité E convertit des formules du second ordre en formules du premier ordre. En outre, une telle transformation fait apparaître des quantificateurs. Enfin, la formule obtenue est bien du premier ordre.

Un détail à noter est qu'au cours de la transformation les symboles représentant les ensembles disparaissent et laissent place à un symbole de prédicat du même nom. Je l'omets par la suite mais il faudrait toujours préciser que les symboles sont choisis de manière à ce que les conflits soient évités.

Enfin remarquons que cette façon de procéder impose des restrictions sur le langage des formules que l'on souhaite manipuler. En particulier, on ne peut pas considérer des ensembles d'ensembles. Par exemple la transformation suivante

ne termine pas sur une formule du premier ordre :

$$
\begin{array}{l}
C = \{A, B\} \text{ avec } A \text{ et } B \text{ des ensembles} \\
C = \{\lambda x.\, x = A \vee x = B\} \\
\forall y.\, \left[\breve{C}(y) = \left(\lambda x.\, x = \breve{A} \vee x = \breve{B}\right)(y)\right] \\
\forall y.\, \left[\breve{C}(y) = (y = \breve{A} \vee y = \breve{B})\right]
\end{array}
$$

Cette dernière formule n'est pas du premier ordre car on quantifie sur y qui est un prédicat.

3.2 Langage d'entrée

Le langage que nous considérons est identique à la logique du première ordre classique avec égalité, augmentée de constructions pour représenter des ensembles :

- $\mathcal{V}$ désigne l'ensemble des noms de variables (Habituellement a,b,c).
- L'ensemble des symboles de fonction est $\mathcal{F}$. Les termes sont de la forme $f(x_1, \ldots, x_n)$ où $x_1, \ldots, x_n$ sont des termes avec $f \in \mathcal{F}$ et f d'arité n.
- L'ensemble des symboles de prédicats est $\mathcal{P}$. Les prédicats sont de la forme $P(x_1, \ldots, x_n)$ où $x_1, \ldots, x_n$ sont des termes et $P \in \mathcal{P}$ et P d'arité n.
- $\mathcal{E}$ désigne l'ensemble des noms d'ensembles (Habituellement A, B, E).

Remarques : Pour que le langage respecte les contraintes souhaitées, on impose que $\mathcal{V}$ et $\mathcal{E}$ soient disjoints. Dans un souci de clarté, on impose également que $\mathcal{P}$ et $\mathcal{E}$ soient disjoints.

Définition 3.2 (Grammaire des formules du premier ordre « FOL »)**.** *Le langage des formules FOL est défini par la grammaire suivante :*

$$B \quad ::= \quad \forall x.\, B \mid \exists x.\, B \mid P \mid t_1 = t_2 \mid B \wedge B \mid B \vee B \mid B \longrightarrow B \mid \neg B$$

où t_1 et t_2 parcourent l'ensemble des termes. P parcourt l'ensemble des prédicats et x parcourt $\mathcal{V}$.

Définition 3.3 (Grammaire des formules ensemblistes « SET »)**.** *Le langage des formules FOL est défini par la grammaire suivante :*

$$
\begin{array}{lll}
B & ::= & v \in E \mid E \subseteq E \mid E \subset E \mid E = E \\
 & ::= & F \mid B \wedge B \mid B \vee B \mid B \longrightarrow B \mid \neg B \\
E & ::= & e \mid E \cap E \mid E \cup E \mid E \setminus E \mid \{t_1, \ldots, t_n\} \mid \emptyset \mid \Omega
\end{array}
$$

où F parcourt les formules FOL, e parcourt $\mathcal{E}$, v parcourt $\mathcal{V}$ et où $t_1, \ldots, t_n$ parcourent l'ensemble des termes.

Remarque : Il n'est pas possible de quantifier sur des ensembles. Ainsi dans $\forall x.\, B$ et $\exists x.\, B$, il faut remarquer que x désigne un nom de variable ($x \in \mathcal{V}$). De même l'expression $\{t_1, \ldots, t_n\}$ du terminal E empêche de former un ensemble d'ensembles comme $\{A, B, C\}$ avec $A, B, C \in \mathcal{E}$ (car $t_1, \ldots, t_n$ sont des termes)[3].

[3] D'où l'intérêt que $\mathcal{V}$ et $\mathcal{E}$ soient disjoints.

On peut remarquer que le symbole = est utilisé avec des arguments différents. Il peut donc représenter une égalité entre termes ou une égalité entre ensembles.

3.3 Propriétés

Lemme 3.4 (terminaison du système de réécriture). *Soit F une formule obtenue grâce à la grammaire 3.2. Alors l'application du système de réécriture de la définition 3.1 à F termine.*

Démonstration : Considérons la fonction σ qui à une formule F associe le nombre de symboles $\in, \emptyset, \Omega, \{\ldots\}, \cap, \cup, \subseteq, \subset$ et $\setminus$ qu'elle contient, on a :
Pour toute règle $R : l \longrightarrow r$ du système de réécriture de la définition 3.1, $\sigma(l) > \sigma(r)$. □

Théorème 3.5 (bon comportement de la transformation). *Soit F une formule formée à l'aide de la grammaire 3.3.*

(i) La formule F' obtenue en réécrivant F à l'aide du système de la définition 3.1 termine et est un λ-terme bien formé.

(ii) L'application de la β-réduction et de l'axiome d'extensionnalité sur F' termine sur une formule F'' qui est une formule du premier ordre.

Démonstration : cf. paragraphe A.1 de l'annexe □

3.4 Définitions

Nous décrivons ici rapidement les définitions nécessaires pour pouvoir parler de la satisfaisabilité d'une formule. On considère le langage des formules du premier ordre décrit en 3.2. Comme le but de la manipulation est de transformer les formules appartenant à SET dans des formules équivalentes de FOL, on ne définit ces notions (en reprenant [GL01]) que pour FOL.

Définition 3.6 (Interprétation et affectation). *Une interprétation I est un ensemble non vide D_I, appelé domaine de l'interprétation, muni :*

(i) d'une application I(f) de D_I^m vers D_I pour chaque symbole de fonction f d'arité m.

(ii) d'une application I(P) de D_I^m vers $\{\top, \bot\}$ pour chaque symbole de prédicat P d'arité m.

Une affectation ρ est une application de $\mathcal{V}$ vers D_I.

Définition 3.7 (Sémantique)**.** *La sémantique des formules du premier ordre FOL est définie de la manière habituelle :*

(i) Pour toute affectation ρ, $\rho[v/x]$ est l'affectation envoyant chaque variable y autre que x vers $\rho(y)$ et x vers v.

(ii) Dans une interprétation I, et modulo l'affectation ρ, la sémantique des termes et des formules est définie par :
- $[\![x]\!]I\rho = \rho(x)$
- $[\![f(t_1, \ldots, t_n)]\!]I\rho = I(f)([\![t_1]\!]I\rho, \ldots, [\![t_n]\!]I\rho)$
- $[\![P(t_1, \ldots, t_n)]\!]I\rho = I(P)([\![t_1]\!]I\rho, \ldots, [\![t_n]\!]I\rho)$
- $[\![\neg P]\!]I\rho = \bot$ *si* $[\![P]\!]I\rho = \top, \bot$ *sinon.*
- $[\![P \wedge Q]\!]I\rho = [\![P]\!]I\rho$ *et* $[\![Q]\!]I\rho$
- $[\![P \vee Q]\!]I\rho = [\![P]\!]I\rho$ *ou* $[\![Q]\!]I\rho$
- $[\![P \longrightarrow Q]\!]I\rho = \neg[\![P]\!]I\rho$ *ou* $[\![Q]\!]I\rho$
- $[\![t_1 = t_2]\!]I\rho = [\![t_1]\!]I\rho$ *égal à* $[\![t_2]\!]I\rho$
- $[\![\forall x.\phi]\!] = \bigwedge_{v \in D_I} [\![\phi]\!]I(\rho[v/x])$
- $[\![\exists x.\phi]\!] = \bigvee_{v \in D_I} [\![\phi]\!]I(\rho[v/x])$

Définition 3.8 (Validité et satisfaisabilité)**.**

*(i) Une formule est **valide** si elle vraie dans tout couple (interprétation, affection).*

*(ii) Une formule est **satisfaisable** si il existe une interprétation et une affectation où elle est vraie.*

3.5 Réécriture de l'arbre syntaxique

Dans un premier temps, nous considérons le cas général, c'est-à-dire qu'on s'intéresse à des λ-termes représentés sous formes d'arbres et en particulier à la taille des arbres manipulés. En effet il n'est pas évident – a priori – de savoir comment la taille de l'arbre va se comporter avec les transformations effectuées. Par exemple la réduction suivante, qui ne fait intervenir que la β-réduction,

$$\begin{array}{lll} & (\lambda p.\, p \vee p\,)\big((\lambda p.\, p \vee p)\,\phi\big) & \\ \succcurlyeq_\beta & \quad (\lambda p.\, p \vee p)\,\phi \vee \phi & \\ \succcurlyeq_\beta & \quad\; \phi \vee \phi \vee \phi \vee \phi & \text{termine sur un arbre où la formule} \\ & & \phi \text{ apparaît désormais quatre fois...} \end{array}$$

Si on suppose que la formule ϕ est un arbre de taille n, cet exemple montre qu'en seulement 2 β-réductions, on obtient un arbre de taille $2^2 * n$ (alors que l'arbre initial avait une taille égale à $n+8$). La taille de l'arbre syntaxique peut donc augmenter après β-réduction.

Plus généralement, à partir d'un arbre de taille $(4 * m) + n$ on peut obtenir un arbre de taille $2^m * n$:

$$\begin{array}{lll} & \overbrace{(\lambda p.\, p \vee p)\big((\lambda p.\, p \vee p)\ \ldots\ (\lambda p.\, p \vee p)\,\phi\big)}^{(\lambda p.\, p \vee p) \text{ apparaît } n \text{ fois}} & \\ \succcurlyeq_\beta^* & \qquad\qquad \phi \vee \cdots \vee \phi & \text{ici, } \phi \text{ apparaît } 2^n \text{ fois...} \end{array}$$

Dans le cas général, la β-réduction a un comportement exponentiel. Nous avons conjecturé que pour le fragment auquel on se restreint, on a un comportement quadratique (voir paragraphe A.2 en annexe). C'est à dire que si on a une

formule SET de taille n, la taille de la formule FOL obtenue après réduction est $O(n^2)$.

4 Reconstruction de preuves

4.1 Description générale

Dans l'exemple du paragraphe 3.1, la transformation des formules ensemblistes en formules du premier ordre fait apparaître des quantificateurs. Or la procédure décrite au paragraphe 2.3 permet uniquement de reconstruire des preuves obtenues grâce à l'algorithme de clôture de congruence. Pour pouvoir décider la validité des formules contenant des quantificateurs il est nécessaire de compléter cette procédure. En premier lieu, il faut savoir comment haRVey décharge les obligations de preuves de formules contenant des quantificateurs :

1. La formule est mise en forme normale négative (NNF), c'est à dire qu'on pousse les négations le plus loin possible vers les atomes[4].
2. Les quantificateurs existentiels sont retirés en utilisant la technique de skolemisation.
3. Les quantificateurs universels sont transformés en une conjonction de littéraux. Intuitivement on applique la sémantique du symbole $\forall$ (cf. définition 3.7).

La troisième assertion est justifiée par le théorème de Herbrand, en particulier cette formulation, dérivée de [Fon04] :

Théorème 4.1 (Herbrand). *Soit ϕ une formule de la forme $\forall x_1 \dots x_n.\psi$, où ψ est sans quantificateur. Alors ϕ est insatisfaisable si et seulement s'il existe un entier k, et k instances[a] closes $\psi\sigma_1, \dots, \psi\sigma_n$ telles que $\psi\sigma_1 \wedge \dots \wedge \psi\sigma_n$ soit insatisfaisable.*

[a] Les instances sont définies de la manière usuelle.

L'algorithme implémenté dans haRVey hérite directement de ce théorème. Quand haRVey reçoit une formule F'' de la logique du premier ordre, il supprime les quantificateurs existentiels par skolemisation (on obtient F'). Ensuite les quantificateurs universels sont remplacés par des conjonctions d'instances closes. A ce moment de la procédure on manipule une formule F qui ne contient que des symboles de fonctions et de prédicats non interprétés et des égalités, mais qui ne contient plus de quantificateurs (cette formule appartient à QF-FOL). L'algorithme de clôture de congruence peut alors être utilisé pour décider la satisfaisabilité de F. Bien sûr comme la logique du premier ordre est indécidable cette procédure peut ne pas terminer. En effet il n'y a pas de limites sur la taille des conjonctions que l'on doit rajouter à la formule.

En fait la procédure terminera si la formule d'entrée est insatisfaisable. On ne peut pas savoir si elle terminera dans le cas contraire. Notons qu'en imposant qu'il n'y ait pas de symbole de fonction sous la portée d'un quantificateur, on obtient facilement un fragment décidable. L'algorithme est le suivant :

[4] En particulier, on considère pour simplifier que le fragment $\forall$FOL, même s'il contient le symbole $\neg$ ne permet pas d'écrire des quantificateurs existentiels. En effet dans ce fragment, une fois la formule mise en forme NNF, celle-ci ne doit contenir que des quantificateurs universels.

prouve F'' **:**
  F' := skolem F''
  n := 0
  res := « »
    tant que res $\neq$ « unsat » **et** res $\neq$ « sat » **:**
      F := instanciations (n, F')
      res := clôture de congruence F
      n := instanciations suivantes (n, F')
      si res == « sat » **et** n == 0
          alors res := « sat »
      si res == « sat » **et** n $\neq$ 0
          alors res := « ne sais pas »

Grâce à la méthode « skolem » , on retire les quantificateurs existentiels. Avec la méthode « instanciations » , on obtient une formule où les quantificateurs universels ont été remplacés par des conjonctions d'instances closes (mais le domaine des quantificateurs n'a peut-être pas été représenté entièrement). Avec « instanciations suivantes » , on récupère le nombre de conjonctions d'instances closes à rajouter lors de la prochaine itération (et si cette méthode renvoie zéro, on sait que la procédure va terminer).

On voit que l'algorithme termine si l'insatisfaisabilité a été dérivée ou si la satisfaisabilité a été dérivée et qu'il n'y plus d'instanciations à faire. Cette dernière procédure nous permet donc de résoudre des formules du premier ordre. Combinée avec la technique décrite à la section 3, on a une méthode pour résoudre des formules ensemblistes. Dans ce qui suit, je décris l'implantation de la reconstruction de ces techniques au sein d'Isabelle/HOL.

En résumé, pour déterminer la validité des formules ensemblistes, il faut donc effectuer les étapes suivantes :

1. Traduire la formule ensembliste d'entrée (appartenant à SET) dans une formule FOL.
2. Mettre la formule obtenue sous forme NNF
3. Éliminer les quantificateurs existentiels.
4. Éliminer les quantificateurs universels (en les instanciant).
5. Résoudre la formule obtenue avec l'algorithme de clôture de congruence (abrévié en CC dans les figures qui suivent).

Nous rappelons que le but est de faire la recherche de preuve avec l'outil automatique haRVey et de certifier le résultat grâce à Isabelle/HOL. haRVey est capable de faire toutes ces opérations. Cependant pour pouvoir vérifier le résultat dans Isabelle/HOL, il faut pouvoir rejouer toutes les étapes. Or comme nous allons l'expliquer dans la suite, il n'est pas utile de tout déléguer à haRVey.

4.2 Reconstruction pour ∀FOL

Dans un premier temps, nous nous sommes intéressés à reconstruire les preuves pour des formules contenant uniquement des quantificateurs universels (fragment ∀FOL). Le but est donc de trouver la preuve dans haRVey puis de la rejouer dans Isabelle/HOL comme résumé sur la figure 4.

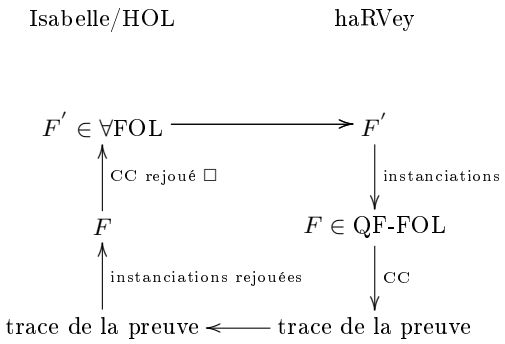

FIG. 4 – Reconstruction pour ∀FOL

Le problème principal est la structure de la formule. Plaçons nous dans le cadre d'un prouveur de théorèmes interactif. Une preuve consiste à déclarer des hypothèses et un but puis appliquer des règles du calcul des séquents pour montrer que les hypothèses impliquent le but. Les règles du calcul des séquents d'Isabelle/HOL est très fourni. En effet même si tout repose sur plus d'une douzaine de règles « de base » , une librairie conséquente existe. Il est donc possible d'appliquer des règles de haut niveau. Cependant ces règles s'appliquent toujours sur le symbole de tête de notre formule. Imaginons par exemple qu'on veuille dériver la validité de :

$$a = b \wedge \big((P \wedge \neg Q) \vee f\ a \neq f\ b\big) \vdash \bot \quad (3)$$

[5]

On voit qu'il faut appliquer la règle de congruence $\dfrac{\Gamma, x = y \vdash \Delta}{\Gamma, f\ x = f\ y \vdash \Delta}$ à a et b. Pour cela on doit déjà s'occuper de la structure booléenne de la formule. Il faut éliminer la conjonction qui est le symbole de tête puis faire une preuve par cas sur la disjonction. En calcul des séquents (écrit de manière usuelle, à l'envers) on obtient :

$$\dfrac{\dfrac{a = b; P \wedge \neg Q \vdash \bot \qquad a = b; f\ a\ \neq\ f\ b \vdash \bot}{a = b; (P \wedge \neg Q) \vee (f\ a\ \neq\ f\ b) \vdash \bot}\ \vee_e}{(a = b) \wedge \big((P \wedge \neg Q) \vee (f\ a\ \neq\ f\ b)\big) \vdash \bot}\ \wedge_e$$

Seulement après ces étapes, on peut appliquer la règle de congruence dans la formule de droite et conclure.

Ainsi on pourrait penser que pour éliminer les quantificateurs (par exemple remplacer $\forall x.x = a$ par $a = a \wedge b = a$ si le domaine est $\{a, b\}$), on a besoin de gérer la structure de la formule étudiée. En réalité, c'est inutile. En effet une fois que la formule n'a plus de quantificateurs universels (c'est-à-dire aucun quantificateur car on s'intéresse à ∀FOL) on utilise un mécanisme d'abstraction booléenne comme décrit au paragraphe 2.2 (ce mécanisme est implanté dans Isabelle/HOL , cf. [Web05]). On peut donc faire les choses de manière très simple. Il suffit de compléter la méthode d'abstraction pour qu'elle prenne en

[5] Ici et dans la suite du document, les formules sont notées avec la syntaxe d'Isabelle/HOL où il n'y a pas de parenthèses même pour les fonctions d'arités supérieures à 2. C'est-à-dire que $f(x, y, z)$ est noté $f\ x\ y\ z$.

compte les quantificateurs universels et *désucrer* les quantificateurs de cette manière :

Supposons que la formule F contient le littéral suivant : $\forall x.\, P\, x$. Notons que ce littéral peut apparaître n'importe où, pas seulement en dessous du symbole de tête. Pour que l'abstraction booléenne prenne en compte les quantificateurs, il nous suffit de rajouter la conjonction $\bigwedge_{v \in D_P} (\forall x.\, P\, x) \longrightarrow P\, v$ à la formule F. L'abstraction booléenne faite ensuite nous garantit que ce raisonnement est correct.

Exemple :

$$a \neq b \wedge \big((P \wedge \neg P) \vee \forall x.\, x = a\big) \qquad \text{devient}^{6}$$

$$\underbrace{a \neq b}_{\neg A} \wedge \big((P \wedge \neg P) \vee \underbrace{\forall x.\, x = a}_{B}\big) \wedge$$
$$(\underbrace{\forall x.\, [x = a]}_{B} \longrightarrow \underbrace{a = a}_{C}) \wedge (\underbrace{\forall x.\, [x = a]}_{B} \longrightarrow \underbrace{b = a}_{A})$$

Comme on l'attendait la formule est insatisfaisable car l'abstraction booléenne ne fournit aucun modèle abstrait. En effet ni

$$\neg A \wedge (P \wedge \neg P) \wedge (B \longrightarrow C) \wedge (B \longrightarrow A)$$

ni

$$\neg A \wedge B \wedge (B \longrightarrow C) \wedge (B \longrightarrow A)$$

ne sont propositionnelement satisfaisables.

Notons que cette procédure ne termine pas toujours, car le nombre des conjonctions à ajouter aux formules pour remplacer les quantificateurs peut être infinie. Mais comme mentionné au paragraphe 4, un algorithme de type *refine and check* permet de garantir la terminaison si la formule d'entrée est insatisfaisable.

4.2.1 En pratique ?

Le format de trace de preuve décrit au paragraphe 2.3 ne convient plus. En effet il est seulement constitué d'une suite de séquences de la forme :

```
but
règle, terme
règle, terme
règle, terme
```

où les règles sont les mots clés TRANS, CONGR, PRED, INEQ ou CONFL et où les termes et le but sont des chaînes de caractères (cf. 2.3).

Désormais il faut rajouter les théorèmes correspondants aux instanciations, dans notre exemple $\forall x.\, [x = a] \longrightarrow a = a$ et $\forall x.\, [x = a] \longrightarrow a = b$. Pour des raisons de performances, la trace de la preuve n'est plus un simple fichier texte (qu'il faut analyser syntaxiquement « à la main » dans le code de notre

[6] Notons que l'univers de Herbrand est $\{a, b\}$.

interface) mais un fichier contenant du code ML, le langage d'implémentation d'Isabelle/HOL. Cela ne demande plus d'analyser le fichier ligne par ligne, il suffit de le charger avec l'interpréteur ML à la volée. De plus, de cette manière on obtient directement un objet ML représentant la trace de la preuve. Enfin, les formules ne sont plus des chaînes de caractères mais sont directement écrites dans le type ML représentant les formules dans Isabelle/HOL. La trace de la preuve est le code ML suivant :

```
(
        rv_proof_congr := ...
        rv_proof_inst := ...
)
```

La variable *rv_proof_congr* reproduit le schéma (but, (règle, terme)∗) sauf que les termes sont écrits en ML directement.

La variable *rv_proof_inst* contient toutes les indications pour rejouer les instanciations. Son type ML est (Term.term ∗ Term.term) list où *Term.term* est le type des formules en Isabelle/HOL. Par exemple, pour montrer la validité de

$$a \neq b \vee f\,a = b \vee g\,b \neq f a \vee \exists x.\, b \neq g\,x$$

on vérifie l'insatisfaisabilité de sa négation :

$$a = b \wedge f\,a \neq b \wedge g\,b = f a \wedge \forall x.\, b = g\,x$$

Les instanciations sont les suivantes :

1. $\forall x.\, [b = g\,x] \longrightarrow b = g\,g\,b$
2. $\forall x.\, [b = g\,x] \longrightarrow b = g\,f\,a$
3. $\forall x.\, [b = g\,x] \longrightarrow b = g\,a$
4. $\forall x.\, [b = g\,x] \longrightarrow b = g\,b$

ce qui se traduit dans la trace de preuve par le code ML visible ci-après. La quatrième instanciation est écrite, directement en ML. On « voit » le quantificateur $\forall x.\, [b = g\,x]$ en premier : (Const("All", T) $ Abs(x, T, Const("eq", boolT) $ Free("b", T) $ Free("g", T -> T)) $ Bound 0, puis le terme $g\,b$ à instancier : (Free("g", T ⟶ T) $ Free("b", T)).

```
rv_proof_inst :=
[
          ...
          (Const(``All'', T)
          $ Abs(x, T, Const(``eq'', boolT)
          $ Free(``b'', T) $ Free(``g'', T --> T)) $ Bound 0,
          Free(``g'', T --> T) $ Free(``b'', T))
          ...
]
```

4.2.2 Une implémentation naïve

Une des réflexions qui, de manière générale, est souvent revenue lors de mon travail a été l'opposition entre une approche laxiste et une approche dirigiste de

la reconstruction. En effet, la librairie de Isabelle est très fournie. Elle permet de faire de l'unification d'ordre supérieur par exemple. Quand on prouve les théorèmes correspondant aux instanciations, on peut soit laisser Isabelle/HOL s'occuper des détails (trouver les termes, les unifier) (approche laxiste) soit tout expliciter (approche dirigiste). Quand on prouve un théorème d'instanciation, on utilise la règle *spec* qui a cette forme :

$$\frac{\Gamma, P\ a \vdash \Delta}{\Gamma, \forall x. P\ x \vdash \Delta}\ spec$$

Par exemple si le théorème d'instanciation est $\forall x. x = a \vdash b = a$, il suffit d'appliquer la règle *spec* et Isabelle/HOL se charge de faire les substitutions $\{\langle P, \lambda x.\ x = a\rangle, \langle y, b\rangle\}$. Cependant cette opération a un coût de calcul. De plus, on peut avoir des cas plus compliqués, par exemple si le séquent qu'on manipule est $\Gamma, \forall x.\ [x = a] \vdash \Delta$. Dans ce cas, Γ peut également contenir des formules quantifiées et Isabelle/HOL échouera à unifier ces littéraux avec le but, avant de trouver le littéral voulu : $\forall x.\ x = a$.

On peut remédier à cela en instanciant explicitement la règle *spec*. En effet, Isabelle/HOL permet de dériver de nouvelles règles à partir de règles existantes et contenant des variables libres. Ainsi pour prouver $\forall x.\ [x = a] \vdash b = a$, on particularise le théorème *spec* en explicitant les substitutions. La règle *spec* permet d'obtenir la règle suivante :

$$\frac{\Gamma, b = a \vdash \Delta}{\Gamma, \forall x. x = a \vdash \Delta}\ spec\ particulier$$

On notera qu'en particularisant *spec* on dérive exactement la règle souhaitée. Appliquer cette règle une fois nous donne le théorème voulu sans reconnaissance de motifs.

4.2.3 Épilogue

Dans le cadre d'un travail de plus longue durée, les pistes suivantes pourraient être suivies :

D'un point de vue technique, la trace de la preuve pourrait être réduite de manière conséquente en faisant des listes d'instanciations. C'est-à-dire remplacer

1. $\forall x.\ b = g\ x,\ g\ b$
2. $\forall x.\ b = g\ x,\ f\ a$

par :

1. $\forall x.\ b = g\ x,\ [g\ b, f\ a]$

Cela se traduirait au niveau de l'objet de preuve par le type ML (Term.term $*$ (Term.term list)) list. Mais ceci est un peu compliqué à faire au niveau d'haRVey.

D'un point de vue théorique, on pourrait améliorer la recherche de preuve en ajoutant des heuristiques lors de la procédure d'instanciation (voir par exemple [HRCS02]). En effet pour le moment la procédure énumère simplement l'univers de Herbrand. Mais on pourrait instancier en premier lieu les quantificateurs qui peuvent amener une inconsistance plus rapidement (car nous sommes intéressés

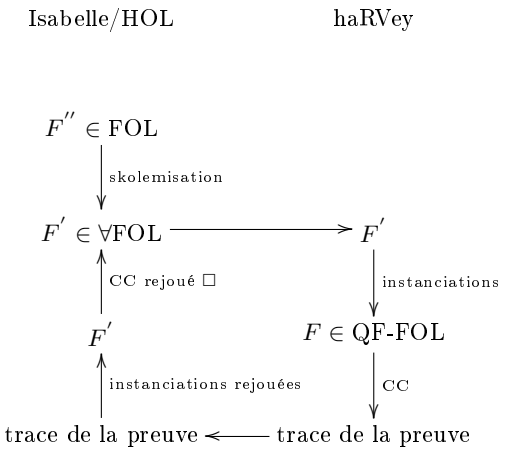

FIG. 5 – Procédure pour FOL

par l'insatisfaisabilité). Imaginons par exemple que l'algorithme de clôture de congruence finisse sur le graphe suivant :

$$a \text{ ——— } b$$

$$f\,a \text{ ——— } f\,b \text{ ——— } g\,c$$

De plus imaginons qu'il faille instancier la formule « $\forall x.x = gc \longrightarrow x \neq f\,b$ ». Le membre droit de $\longrightarrow$ a une polarité négative car c'est une dis-égalité. Si on considère les égalités comme positives, dériver une contradiction consiste à trouver un même littéral apparaissant positivement et négativement dans la formule. Dans notre exemple le littéral $x = f\,b$ apparaît positivement (modulo la symétrie de l'égalité) dans $f\,a = f\,b$ et $f\,b = g\,c$. On voit donc qu'instancier le quantificateur avec $f\,a$ ou $g\,c$ serait intéressant.

4.3 Procédure pour FOL

Comme visible sur la figure 5, on souhaite cette fois reconstruire des preuves pour des formules du premier ordre sans restrictions, c'est-à-dire contenant des quantificateurs existentiels et universels. Il faut soit reconstruire l'algorithme de skolemisation de haRVey soit faire la skolemisation directement dans Isabelle/HOL, et envoyer la formule à haRVey ensuite. Dans le paragraphe suivant, nous décrivons rapidement les différentes techniques de skolemisation puis expliquons les choix que cela nous a amené à faire.

4.3.1 Différentes skolemisations

La technique de skolemisation est souvent présentée sous sa forme la plus simple, la skolemisation naïve. Dans ce cas, la formule est mise sous forme prénexe : on « pousse » tous les quantificateurs devant la formule pour obtenir

une formule de la forme $Q_1 \dots Q_n.\ \phi$ où chaque Q_i est soit $\forall$ soit $\exists$ et où ϕ ne contient pas de quantificateur. Cette méthode est naïve car elle introduit des symboles de skolems inutiles ou ces symboles ont une arité trop grande. Par exemple la forme prénexe de

$$\forall x.\ \big[(\forall y.\exists z.P\ x\ y\ z)\ \wedge\ (\forall u.\exists w.Q\ x\ u\ w)\big] \tag{4}$$

est

$$\forall x \forall y \exists z \forall u \exists w.(P\ x\ y\ z\ \wedge\ Q\ x\ u\ w) \tag{5}$$

On obtient, après skolemisation :

$$\forall x \forall y \forall u.(P\ x\ y\ (f\ x\ y)\ \wedge\ Q\ x\ u\ (g\ x\ y\ u)) \tag{6}$$

On s'aperçoit que la skolemisation de w crée un symbole de skolem g d'arité 3. Or ceci est surprenant car dans la formule initiale (4), w ne dépend que de u et x.

Pour remédier à cela, plusieurs autres techniques de skolemisation ont été introduites, en particulier la skolemisation interne. Cette technique consiste à minimiser les dépendances des variables skolemisées. Ainsi le résultat de la skolemisation interne de (4) est :

$$\forall x.\ \big[(\forall y.P\ x\ y\ (f\ x\ y))\ \wedge\ (\forall u.Q\ x\ u\ (g\ x\ u)\big] \tag{7}$$

haRVey effectue une skolemisation interne. Pour pouvoir la rejouer dans Isabelle/HOL il faut définir les notions utilisées (voir [Non96] pour plus de précisions) :

Définition 4.2 (Occurrence d'un quantificateur)**.** *Par simplicité on suppose que, quelle que soit la formule ϕ considérée, tous les quantificateurs apparaissant dans ϕ lient une variable avec un nom différent*[a]*.*

(i) On note $\mathcal{V}(\phi)$ l'ensemble des variables libres (non liées par un quantificateur) dans une formule ϕ.

(ii) Pour chaque variable x apparaissant dans ϕ, l'unique sous-formule commençant par un quantificateur qui lie x est noté ϕ^x. Ainsi si $\exists x.\psi\ x$ est une sous-formule de ϕ, $\phi^x = \exists x.\psi\ x$.

[a] Ceci permet de désigner une sous-formule commençant par un quantificateur sans ambiguïté (chaque occurrence d'une sous-formule de la forme $\exists x.\phi\ x$ ou $\forall y.\psi\ y$ étant unique).

Définition 4.3 (Skolemisation interne)**.** *La skolemisation interne d'une formule ϕ consiste à remplacer chaque occurrence de sous-formules de la forme $\exists x.\psi\ x$ dans ϕ par $\psi\{x/f\ y_1 \dots y_n\}$ où :*

(i) f est un nouveau symbole de fonction.

(ii) $\{y_1, \dots, y_n\} = \mathcal{V}(\exists x.\psi\ x)$, autrement dit l'ensemble des variables libres de la sous-formule $\exists x.\psi\ x$.

Rejouer la skolemisation interne effectuée par haRVey pour obtenir une formule $\forall$FOL à partir d'une formule FOL nécessite en particulier de formaliser la notion d'ensemble de variables libres dans Isabelle/HOL. Dans le paragraphe 4.3.2, nous présentons les conséquences de cette remarque.

Notons que cette étape est correcte car le fait de skolemiser une formule ne change pas sa satisfaisabilité, comme exprimé dans le théorème suivant :

Théorème 4.4 (La skolemisation préserve la satisfaisabilité). *Soit F une formule et F' la formule obtenue en skolemisant F. Alors F' est satisfaisable si et seulement si F est satisfaisable.*

Démonstration : Voir [Fon04]. □

4.3.2 Contraintes pour la reconstruction

Dans Isabelle/HOL la skolemisation est possible grâce à la règle suivante :

$$\frac{\Gamma, \exists f. \forall x. P\ x\ (f\ x) \vdash \Delta}{\Gamma, \forall x. \exists y. P\ x\ y \vdash \Delta}\ choice$$

On voit que cette règle permet d'éliminer une variable quantifiée existentiellement, en introduisant une nouvelle variable quantifiée existentiellement, mais se trouvant à l'extérieur du quantificateur universel considéré. Combinée avec la règle d'élimination de quantificateur existentiel, on peut enlever tous les quantificateurs existentiels d'une formule, comme ici[7] :

$$\cfrac{\cfrac{\cfrac{\Gamma, \forall x. P\ \wedge\ Q\ x\ (f\ x) \vdash \Delta}{\Gamma, \exists f. \forall x. P\ \wedge\ Q\ x\ (f\ x) \vdash \Delta}\ \exists\text{ elimination}}{\Gamma, \forall x. \exists y. P\ \wedge\ Q\ x\ y \vdash \Delta}\ \text{choice}}{\Gamma, \forall x. P\ \wedge\ \exists y. Q\ x\ y \vdash \Delta}\ \text{extrascope}\ \exists$$

Fig. 6 – Exemple de skolemisation en Isabelle/HOL

Ce comportement semble correspondre à la définition de la skolemisation interne. Malheureusement, la règle *choice* a un effet local : combiné avec $\exists$ *elimination*, elle permet d'obtenir $\forall x. P\, x\, (f\, x)$ à partir de $\forall x. \exists y. P\, x\, y$. Mais elle ne permet pas de prendre en compte les variables libres autre que x apparaissant dans $\exists y. P\, x\, y$. Autrement dit impossible de formaliser le point 2 de la définition de la skolemisation interne (voir 4.3). En fait pour skolemiser une sous-formule $\exists x. \phi\ x$ dont l'ensemble des variables libres est $\{y_1, \ldots, y_n\}$, il faut appliquer la règle *choice* un nombre n de fois, comme dans l'exemple suivant :

$$\cfrac{\cfrac{\cfrac{\Gamma, \forall x. \forall y. P\ x\ y\ (g\ x\ y) \vdash \Delta}{\Gamma, \exists g. \forall x. \forall y. P\ x\ y\ (g\ x\ y) \vdash \Delta}\ \exists\text{ elimination}}{\Gamma, \forall x. \exists f. \forall y. P\ x\ y\ (f\ y) \vdash \Delta}\ choice}{\Gamma, \forall x. \forall y. \exists z. P\ x\ y\ z \vdash \Delta}\ choice$$

On s'aperçoit que la règle *choice* ne permet pas d'effectuer une skolemisation interne de la même manière qu'haRVey. En effet la skolemisation interne élimine un quantificateur existentiel en une fois, alors que la règle *choice* supprime uniquement une dépendance de la sous-formule skolemisée. C'est pourquoi simuler la skolemisation interne avec une seule application de *choice* n'est pas possible. Il faut utiliser une méthode différente, comme présentée dans la sous-section suivante.

[7] Notons que l'application de la règle $\exists$ *elimination* est possible car f n'apparaît ni dans Γ, ni dans Δ (en effet l'application de *choice* génère une variable liée, sans nom)

Pour que la skolemisation interne et la skolemisation par application successives de *choice* aient un comportement similaire, il faut auparavant mettre la formule considérée sous une forme appropriée. Il est nécessaire que chaque quantificateur existentiel soit précédé par tous les quantificateurs universels liant des variables dans la sous formule existentiellement quantifiée considérée. Ceci revient à mettre la formule sous forme prénexe : l'avantage de la skolemisation interne est donc perdu...

C'est pourquoi nous avons choisi de skolemiser la formule directement en Isabelle/HOL en appliquant successivement *choice* (voir figure 5). De plus, si on la précède par un *miniscoping*, il semble que cette technique donne des résultats aussi bons[8] que la skolemisation interne.

4.3.3 Épilogue

Nous avons expliqué au paragraphe précédent qu'il n'est pas possible de rejouer une skolemisation interne en appliquant une seule règle Isabelle/HOL. Cependant on peut utiliser les outils automatiques disponibles pour simuler une skolemisation interne. Par exemple la skolemisation de (8) donne (9) :

$$\forall x.P\ x\ \wedge\ \forall y.\exists z.Q\ x\ y\ z \tag{8}$$

$$\forall x.P\ x\ \wedge\ \forall y.Q\ x\ y\ (f\ x\ y) \tag{9}$$

Or expérimentalement, la tactique automatique *fast* arrive à prouver le théorème correspondant :

$$\frac{\vdash \forall x.P\ x\ \wedge\ \forall y.\exists z.Q\ x\ y\ z}{\vdash \forall x.P\ x\ \wedge\ \forall y.Q\ x\ y\ (f\ x\ y)}\ fast$$

Ainsi on pourrait imaginer que haRVey donne les informations nécessaires (la position de la sous-formule à skolemiser et l'ensemble de ses variables libres) à chaque skolemisation et prouver le théorème correspondant (par exemple le théorème précédent) en Isabelle/HOL avec *fast*. Mais cela poserait plusieurs problèmes :

1. La procédure *fast* entièrement automatique peut avoir un comportement inattendu.
2. Sur des formules très grosses elle échouerait car ces tactiques automatiques ne sont pas adaptées.

Or notre but est d'augmenter le degré d'automatisation d'Isabelle/HOL pour ces formules conséquentes. C'est pourquoi la procédure par induction (par applications successives de *choice*) a été choisie.

4.4 Procédure pour SET

Cette fois, on veut faire toute la suite de transformations précédemment expliquées. C'est-à-dire qu'on souhaite traiter des formules appartenant à SET directement dans Isabelle/HOL , toujours en déchargeant la recherche de preuve à haRVey. Rappelons que, comme expliqué au paragraphe 3.1, on va transformer les formules SET en formules FOL puis utiliser les méthodes décrites auparavant

[8] au sens décrit dans [Fon04], c'est à dire qu'une technique de skolemisation est meilleure qu'une autre si elle produit des symboles de skolems d'arités et d'imbrications plus petites.

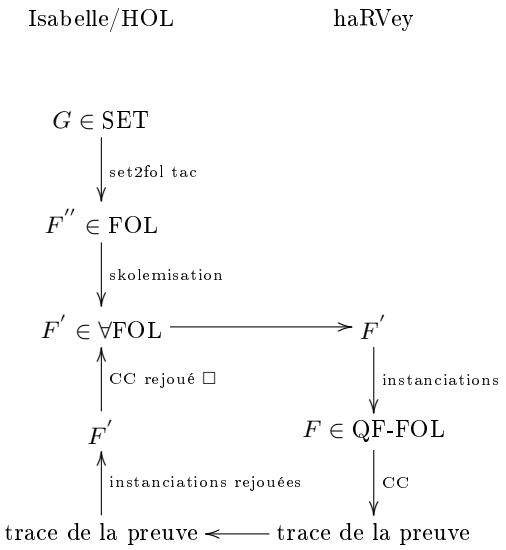

FIG. 7 – Procédure pour SET

pour résoudre les formules FOL.

La méthode que j'ai étudié au paragraphe 3.1 est implémentée dans haRVey. Ceci ajoute la possibilité d'envoyer directement des formules SET à haRVey. Cependant, il ne nous a pas semblé approprié d'utiliser cette fonctionnalité, pour plusieurs raisons :

- D'un point de vue conceptuel, haRVey est un outil de recherche de preuve, pas un « traducteur » de formule.
- Comme on l'a vu dans la reconstruction pour ∀FOL et FOL, rejouer une preuve demande d'avoir défini précisément les notions manipulées (position d'un sous-arbre, skolemisation, instanciations ...). Le fait de réécrire une formule à l'aide des opérations visibles sur la figure 3.1 sera difficile à formaliser et à reproduire en Isabelle/HOL.
- Cette transformation s'applique sur des sous-formules donc, comme expliqué au paragraphe 4.2, cela pose des problèmes car en Isabelle/HOL, on ne peut appliquer les règles que sur le symbole de tête des formules.
- La procédure *divide_and_conquer* décrite dans [Web05, Cha06], simple à implanter, et reflétant le raisonnement naturel, s'applique parfaitement dans notre cas.

Nous avons donc choisi d'effectuer la transformation de formules SET en formules FOL au niveau d'Isabelle/HOL comme résumé sur la figure 7. Cela signifie qu'il faut formaliser l'isomorphisme existant entre les ensembles et les prédicats, puis utiliser cet isomorphisme pour déduire une formule FOL à partir d'une formule SET dans le calcul implanté par Isabelle/HOL.

4.4.1 Diviser pour mieux régner

La procédure *divide_ and_ conquer* consiste à faire une preuve par induction sur la structure des formules. Ainsi on part des axiomes, et pour chaque connecteur logique α on crée le théorème $P \alpha Q$ depuis les preuves de P et Q. Qu'entend t'on ici par « axiomes » ? Il s'agit de l'équivalence entre chaque symbole d'ensemble A et son prédicat associé $\breve{A}$. Ainsi si la formule ensembliste considérée contient les variables d'ensembles suivantes : $A_1, \ldots, A_n$, nos axiomes seront :

$$\begin{aligned} \forall x.\, [(x \in A_1) &= \breve{A}_1\, x] \\ &\vdots \\ \forall x.\, [(x \in A_n) &= \breve{A}_n\, x] \end{aligned}$$

Toutefois, cela n'est pas suffisant, il faut aussi considérer les ensembles qui sont des explicitations (cf. grammaire 3.3). A l'énumération $\{a_1, \ldots, a_n\}$, on associe l'axiome :

$$\forall x.\, [x \in \{a_1, \ldots, a_n\}] = (x = a_1 \ \vee \ \ldots \ \vee \ x = a_n)$$

Il y a deux choses à considérer ici. La structure logique de la formule (connecteurs $=, \wedge, \vee, \longrightarrow, \neg, \in, \subseteq$ et $\subset$) et la structure des ensembles apparaissants dans la formule (connecteurs $\cap, \cup$ et $\setminus$). Dans les deux paragraphes suivants, on présente successivement l'algorithme gérant la structure logique puis l'algorithme gérant la structure ensembliste.

4.4.2 Connecteurs logiques

Notre algorithme reflète directement le processus d'induction sur la structure de la formule. En effet, grâce aux théorèmes Isabelle/HOL qui suivent, on associe une preuve de $P\, \alpha\, Q$ à une preuve de P et une preuve de Q ou α peut être $=, \wedge, \vee, \longrightarrow, \neg, \in, \subseteq$ ou $\subset$. Dans le tableau suivant on trouve les théorèmes Isabelle/HOL pour chaque symbole logique. La première ligne montre les théorèmes pour plusieurs opérateurs (α peut être $=, \wedge, \vee$ ou $\longrightarrow$) :

opérateur	théorème		
α	$P' = P;\ Q' = Q$	$\vdash$	$(P'\, \alpha\, Q') = (P\, \alpha\, Q)$
$\neg$	$P' = P$	$\vdash$	$\neg P' = \neg P$
$\in$	$\breve{A}\, a$	$\vdash$	$a \in A$
$\subseteq$	$\forall x.\, \breve{P}\, x \longrightarrow \breve{Q}\, x$	$\vdash$	$P \subseteq Q$
$\subset$	$(\forall x.\, \breve{P}\, x \longrightarrow \breve{Q}\, x) \ \wedge \ (\exists x.\, \breve{Q} x \ \wedge \ \neg \breve{P} x)$	$\vdash$	$P \subset Q$
$=$	$\forall x.\, \breve{P}\, x = \breve{Q}\, x$	$\vdash$	$A = B)$
ax	P	$\vdash$	P

Dans ce tableau les expressions primées (respec. non primées) représentent les formules FOL (respec. SET). Les deux premières lignes de ce tableau permettent de gérer la structure logique de la formule. On remarquera qu'à droite

du symbole $\vdash$, on a une égalité dont le membre gauche appartient à FOL alors que le membre droit appartient à SET. Les quatres lignes suivantes permettent de prendre en compte les expressions ensemblistes renvoyant une valeur de vérité. Notons que cette fois le symbole $=$ s'appliquent à deux ensembles. Si on ne considère que les symboles $\wedge$, $\neg$ et $\in$, l'algorithme correspondant est visible sur la figure 4.4.2 page 26.

```
divide_and_conquer F :
  case F of P ∧ Q :
    P' := divide_and_conquer P
    Q' := divide_and_conquer Q
    return combine (∧, P', Q')

  case F of ¬P :
    P' := divide_and_conquer P
    return combine (¬, P')

  case F of a ∈ A :
    T := get_axiom A
    return combine (∈, a, T)

  case F of _ :
    return combine (ax, F)
```

FIG. 8 – Algorithme minimal pour passer de SET à FOL

Cet algorithme (dans le cas de $\wedge$) permet de combiner une preuve du membre gauche P et du membre droit Q pour construire une preuve de $P \wedge Q$. Dans le cas d'une expression ensembliste (**case** F **of** $a \in A$), cet algorithme récupère l'axiome (grâce à *get_axiom*) liant l'ensemble atomique A et $\breve{A}$. Notons que le cas par défaut (**case** F **of** _) permet d'appliquer cet algorithme avec n'importe quelle autre théorie (pas seulement SET). Par exemple, sur la formule $P \wedge Q \wedge (1+3<x)$ (où P et Q sont dans SET) il renverra $P' \wedge Q' \wedge (1+3<x)$ (avec P' et Q' dans FOL) car le symbole $<$ sera géré par le cas par défaut.

Exemple :
divide_and_conquer « $b \in \{a,b,c\} \wedge (1+3=5) \wedge (R \vee \neg R)$ » appelle :
divide_and_conquer « $(R \vee \neg R)$ » := $(R \vee \neg R)$
divide_and_conquer « $(1+3<5)$ » := $(1+3<5)$
divide_and_conquer « $b \in \{a,b,c\}$ » := $(b=a \vee b=b \vee b=c)$

Ensuite *combine* crée la règle :

$$\frac{\vdash (b=a \vee b=b \vee b=c) \wedge (1+3=5) \wedge (R \vee \neg R)}{\vdash b \in \{a,b,c\} \wedge (1+3=5) \wedge (R \vee \neg R)}$$

qui est donc le résultat de notre algorithme.

Notons qu'on a en bas la formule initiale, et en haut une formule FOL (modulo l'arithmétique évidemment). Avec cette règle, il suffit donc de faire un pas

de plus à l'envers dans le calcul des séquents d'Isabelle/HOL pour avoir une formule FOL comme nouveau but.

Cet algorithme permet de gérer la structure booléenne de la formule, mais il faut encore transformer les ensembles non atomiques en prédicats comme expliqué dans le paragraphe suivant.

4.4.3 Structure ensembliste

A nouveau on utilise un schéma d'induction sur la structure des ensembles (opérateurs $\cap, \cup, \setminus, \emptyset$ et Ω). On obtient un algorithme (qu'on appelle *set2fol_rule* identique dans la structure à *divide_and_conquer* mais qui prend en paramètre un ensemble). Cet algorithme, à partir des axiomes sur les variables d'ensembles et les ensembles explicites, construit une règle Isabelle/HOL donnant l'équivalence d'un ensemble de la forme $P\,\alpha\,Q$ où α peut être $\cap, \cup, \setminus, \emptyset$ ou Ω avec un prédicat.

Voici les théorèmes Isabelle/HOL utilisés pour faire l'induction :

opérateur	théorème
$\cap$	$\vdash \forall x. \breve{A}\,x \ \wedge\ \breve{B}\,x = x \in (A \cap B)$
$\cup$	$\vdash \forall x. \breve{A}\,x \ \vee\ \breve{B}\,x = x \in (A \cup B)$
$\setminus$	$\vdash \forall x. \breve{A}\,x \ \wedge\ \neg\breve{B}\,x = x \in (A \setminus B)$
$\emptyset$	$\vdash \forall x. \bot = (x \in \emptyset)$
Ω	$\vdash \forall x. \top = (x \in \Omega)$

Si on lui donne en entrée $(\emptyset \cup \{a, b, c\}) \cap A$, cet algorithme renvoie la règle :

$$\frac{\vdash \forall x.\ \left[(\bot \ \vee\ (x = a \ \vee\ x = b \ \vee\ x = c) \ \wedge\ \breve{A}\,x\right]}{\vdash \forall x.\ x \in \big((\emptyset \cup \{a, b, c\}) \cap A\big)}$$

La combinaison de *divide_and_conquer* et *set2fol_rule* permet de passer d'une formule SET à une formule FOL de manière élégante en Isabelle/HOL. Un exemple d'utilisation directement en Isabelle/HOL est visible dans les annexes au paragraphe B.

4.4.4 Une méthode alternative : la réflexion

Dans ce paragraphe, nous présentons une autre implantation permettant d'obtenir le théorème Isabelle/HOL $\frac{\vdash F}{\vdash S}$ où $F \in$ FOL et $S \in$ SET. Cette procédure est largement inspirée de [Cha06].

La réflexion (présentée plus en détail dans [Har95]) consiste à remplacer la recherche de preuve par du calcul : on va représenter (refléter) les formules SET par des formules $\mathcal{S}$. On définit pour cela la fonction *reify* : SET $\longrightarrow \mathcal{S}$. Cette fonction sera récursive et suivra la structure des formules SET. Ensuite on calcule la formule FOL équivalente à la formule appartenant à $\mathcal{S}$ (grâce à la fonction *set2fol*). Soit $(\!|.|\!)$ la sémantique des formules HOL. Pour garantir la correction de ce processus on définit la fonction $[\![.]\!] : \mathcal{S} \longrightarrow \{\top, \bot\}$, autrement dit la sémantique des formules $\mathcal{S}$ et on prouve :

$$\forall S.\ [\![set2fol(S)]\!] = [\![S]\!] \tag{10}$$

Désormais, partant d'une formule SET S', on obtient une formule FOL F, et le théorème $F \vdash S'$ de la manière suivante :

1. On applique *reify* à S' et on obtient S appartenant à $\mathcal{S}$. On prouve que $[\![S]\!] = (\!|S'|\!)$.
2. On applique *set2fol* à S pour obtenir une formule F appartenant à FOL. En instanciant le théorème 10, on a $[\![F]\!] = [\![S]\!]$. Cette étape peut être faite de deux manières : soit en utilisant le simplificateur d'Isabelle/HOL soit en exécutant le code extrait de la définition de *set2fol* (voir [NBS06] pour un exemple d'extraction de code en Isabelle/HOL).

Par transitivité on obtient $[\![F]\!] = (\!|S'|\!)$.

4.4.5 Comparatif

Une fois les deux méthodes (la méthode *divide_and_conquer* qu'on appelle ici LCF puisqu'elle repose sur des tactiques et la réflexion) implantées, nous avons testé le temps que la transformation SET $\longrightarrow$ FOL prenait. La taille des formules est indiquée en nombre de caractères. Le temps est indiqué en secondes.

Les tests ont été faits sur une machine sous Linux, dotée d'un processeur cadencé à 1,6 Ghz. Un « x » indique que la méthode échoue.

test n°	taille	LCF	réflexion
1	3401	2,13	0,72
2	3908	15,69	1,56
3	7071	4,11	2,08
4	4603	4,01	2,40
5	5906	59,12	2,85
6	7388	x	1,02
7	12666	65,30	6,99
8	11722	7,44	2,85

La méthode utilisant la réflexion est au minimum deux fois plus rapide que la méthode LCF. Ceci est assez surprenant, car contraire au résultat de [Cha06]. On peut remarquer que sur les tests 5 et 7, la méthode LCF est bien plus lente que la réflexion. Ces deux exemples ont en commun d'utiliser des ensembles avec un grand nombre d'éléments et un petit nombre de connecteurs. Ceci est dû à l'implémentation de l'algorithme.

Dans le paragraphe 4.4.4, nous avions précisé que la réflexion pouvait être réalisée de deux manières. Ces tests utilisent le simplificateur d'Isabelle/HOL. En extrayant le code des définitions des fonctions définies dans la logique, on aura une amélioration des performances significatives[9]. Un exemple d'utilisation dans Isabelle/HOL est visible dans les annexes au paragraphe B.

[9] Évidemment pour garantir la correction de la méthode, il faut garantir la correction de l'extracteur de code...

5 Conclusion et perspectives

Nous avons présenté dans ce document de nouvelles techniques pour démontrer automatiquement des théorèmes de la logique du premier ordre augmentée de constructions ensemblistes. Ces techniques bénéficient d'un haut degré de confiance car toutes les étapes de la preuve sont certifiées par le noyau d'Isabelle/HOL. De plus, la procédure est entièrement automatique et ne demande pas d'interaction de l'utilisateur.

Nous avons présenté la reconstruction pour des fragments de plus en plus expressifs, successivement les formules du premier ordre sans quantificateur existentiel, les formules du premier ordre puis des formules ensemblistes. L'originalité de ce travail est diverse. En effet, la reconstruction n'avait pas été mise en œuvre pour le fragment étudié. La technique d'instanciation n'avait pas été utilisée pour la reconstruction, nous avons montré qu'elle était particulièrement adaptée. Nous avons mis en évidence la difficulté à rejouer une skolemisation interne dans un prouveur interactif et plus généralement toute étape faisant intervenir une sous-formule. Les techniques présentées pour gérer la structure logique et ensembliste des formules sont facilement réutilisables. La nécessité de suivre une approche dirigiste, où aucun détail n'est laissé au prouveur interactif a été mise en évidence. Sinon, dès qu'un cas particulier apparaîtra, la reconstruction échouera.

L'expression de la trace de preuve directement dans le langage d'implémentation du prouveur interactif est une idée qu'il conviendrait de compléter dans un travail de plus longue haleine. En effet, comme le type des objets dans la trace de preuve encode une information, on peut imaginer qu'il serait possible d'échanger des informations de très haut niveau (exprimer une substitution ou une instanciation par un type ML par exemple). De plus cette technique devrait permettre une augmentation des performances sur des exemples très conséquents. Environ 800 lignes de codes ML ont été écrites pour implémenter les algorithmes en Isabelle/HOL. Le code de l'interface existant avant le début du travail a été considérablement amélioré et simplifié. Enfin, La technique présentée s'étendrait aisément à d'autres théories, comme par exemple l'arithmétique.

A Annexes

A.1 Preuve théorème 3.5

Dans toute la preuve, je ne considère que les cas où la formule fait intervenir un symbole ensembliste.

Prouvons d'abord que la réécriture puis la β-réduction des opérateurs $\cap, \cup, \setminus$ et $\{\ldots\}$ termine sur un λ-terme bien formé qui représente un prédicat (c'est à dire de la forme $\lambda x.\, P(x)$ avec P ne contenant pas de λ-termes). Faisons une preuve par induction sur la structure de la formule :

(i) Soit $E \in \mathcal{E}$ un symbole d'ensemble et $a_1, \ldots, a_n$ des termes. Les ensembles atomiques sont de la forme E, $\{a_1, \ldots, a_n\}$, $\emptyset$ ou Ω. La réécriture termine sur :

(a) $E \longrightarrow \lambda x.\, E(x)$

(b) $\{a_1, \ldots, a_n\} \longrightarrow \lambda x.\, (x = a_1 \vee \cdots \vee x = a_n)$

(c) $\emptyset \longrightarrow \lambda x.\, \bot$

(d) $\Omega \longrightarrow \lambda x.\, \top$

On a des λ-termes bien formés et qui représentent des prédicats.

(ii) Les ensembles non-atomiques sont de la forme $E \,\alpha\, F$ (où α peut être $\cap, \cup$ ou $\setminus$). Par hypothèse d'induction, la réécriture puis la β-réduction de E et F terminent sur des λ-termes de la forme $\lambda y.\, E(y)$ et $\lambda z.\, F(z)$ avec E et F ne contenant pas de λ-termes (on note $\succcurlyeq_\beta^{\rightarrow}$ cette étape). Examinons la réécriture puis la β-réduction de $E \cap F$:

(a) $\begin{aligned} E \cap F &\longrightarrow \big(\lambda PQ.\, \lambda x.\, P(x) \wedge Q(x)\big)\,(E, F) \\ &\succcurlyeq_\beta^{\rightarrow} \big(\lambda PQ.\, \lambda x.\, P(x) \wedge Q(x)\big)\,(\lambda y.\, E(y), \lambda z.\, F(z)) \\ &\succcurlyeq_\beta \lambda x.\, (\lambda y.\, E(y))(x) \wedge\ (\lambda z.\, F(z))(x) \\ &\succcurlyeq_\beta \lambda x.\, E(x) \wedge F(x) \end{aligned}$

qui est bien formé car E et F ne contiennent pas de λ-termes (idem pour $\cup$ et $\setminus$).

Prouvons maintenant que les formules booléennes sont bien des formules du premier ordre après les transformations :

(i) Soient $v \in \mathcal{V}; E, F \in \mathcal{E}$. Les formules atomiques sont de la forme $v \in E, E \subseteq F, E \subset F$, ou $E = F$. Examinons l'effet de la réécriture et de la β-réduction :

(a) $\begin{aligned} v \in E &\longrightarrow \big(\lambda xP.\, P(x)\big)(v, E) \\ &\succcurlyeq_\beta^{\rightarrow} \big(\lambda xP.P(x)\big)(v, \lambda y.E(y)) \\ &\succcurlyeq_\beta E(v) \end{aligned}$

(b) $\begin{aligned} E \subseteq F &\longrightarrow (\lambda PQ.\, \forall x.\big[P(x) \Rightarrow Q(x)\big])(E, F) \\ &\succcurlyeq_\beta^{\rightarrow} (\lambda PQ.\, \forall x.\big[P(x) \Rightarrow Q(x)\big])(\lambda y.E(y), \lambda z.F(z)) \\ &\succcurlyeq_\beta \forall x.\big[\big(\lambda y.E(y)\big)(x) \Rightarrow \big(\lambda z.F(z)\big)(x)\big] \\ &\succcurlyeq_\beta \forall x.\big[E(x) \Rightarrow F(x)\big] \end{aligned}$

(c) $\begin{aligned} E = F &\longrightarrow \big(\lambda PQ.\, \forall x.\big[P(x) = Q(x)\big]\big)(E, F) \\ &\succcurlyeq_\beta^{\rightarrow} \big(\lambda PQ.\, \forall x.[P(x) = Q(x)]\big)(\lambda y.E(y), \lambda z.F(z)) \\ &\succcurlyeq_\beta \forall x.\big[(\lambda y.E(y))(x) = (\lambda z.F(z))(x)\big] \\ &\succcurlyeq_\beta \forall x.\big[E(x) = F(x)\big] \end{aligned}$

Toutes les formules obtenues sont bien des formules du premier ordre (idem pour $\subset$).

(ii) Soient E et F des ensembles. Les formules non-atomiques sont de la forme $v \in E, E \subseteq F, E \subset F$, ou $E = F$. Comme nous l'avons démontré auparavant, la réécriture puis la β-réduction de E et F termine sur des λ-termes qui représentent des prédicats, c'est à dire qu'ils sont de la forme $\lambda y.E(y)$ et $\lambda z.F(z)$.
L'étude de la réécriture puis de la β-réduction revient aux mêmes calculs que dans le cas de base où E et F sont des variables, donc les formules obtenues sont également bien du premier ordre. □

A.2 Conjecture sur la complexité

Les définitions suivantes sont nécessaires :

Définition A.1 (Taille d'une formule). *Pour définir la taille d'une formule, on définit déjà la taille de ses sous-formules :*

(i) La taille de la variable a est $t(a) = 1$.

(ii) La taille du terme $f(t_1, \ldots, t_n)$ (où $t_1, \ldots, t_n$ sont des termes) est $t(a) = t(t_1) + \cdots + t(t_n) + 1$.

(iii) La taille du prédicat $P(t_1, \ldots, t_n)$ (où $t_1, \ldots, t_n$ sont des termes), $t(P) = t(t_1) + \cdots + t(t_n) + 1$.

La taille d'une formule ϕ, notée $t(\phi)$ est définie de la manière suivante :

(i) Les formules de la forme $\neg P$ ont une taille égale à $t(P) + 1$.

(ii) Les formules de la forme $P \alpha Q$ (où α est $\wedge, \vee$ ou $\longrightarrow$) ont une taille égale à $t(P) + t(Q) + 1$.

(iii) Les formules de la forme $\forall x.\phi$ ou $\exists x.\phi$ ont une taille égale à $t(\phi) + 2$.

Définition A.2 (Taille d'une formule ensembliste). *La taille d'un ensemble E notée $t(E)$ est définie de la manière suivante :*

(i) Si E est une variable d'ensemble ($E \in \mathcal{E}$), $t(E) = 1$.

(ii) Si E un ensemble de la forme $E_1 \alpha E_2$ (où α peut-être $\cap, \cup$ ou $\setminus$), sa taille est égale à $t(E_1) + t(E_2) + 1$.

(iii) L'ensemble $\{a_1, \ldots, a_n\}$ (où $a_1, \ldots, a_n$ sont des termes) a une taille égale à $t(a_1) + \cdots + t(a_n) + 1$.

Définition A.3 (Taille d'un λ-terme). *La taille d'un λ-terme M notée $t(M)$ est définie de la manière suivante :*

(i) La taille d'un λ-terme de la forme $\lambda x.\, M$ est égale à $t(M) + 2$.

(ii) La taille d'un λ-terme de la forme $(\lambda x.\, M)N$ est égale à $t(M) + t(N) + 3$.

Si M est λ-terme de la forme $\lambda x.N$, on note $o(M)$ le nombre d'occurrences de x dans N. Par exemple $o(\lambda x.y) = 0, o(\lambda x.x \vee x) = 2$.

Si M est λ-terme de la forme $\lambda x.N$, on note $t^*(M)$ la taille de N. Par exemple $t^*(\lambda x.y) = 1, t^*(\lambda x.x \vee x) = 3$. Ceci est utile dans la suite pour déterminer la taille d'une formule après β-réduction.

En premier, on étudie la transformation des ensembles. Ensuite on étudie les formules booléennes ensemblistes ($v \in E$, $E \subseteq F$, $E \subset F$, $E = F$)

Soit $e \in \mathcal{E}$ un symbole d'ensemble et $a_1, \ldots, a_n$ des termes. Les ensembles atomiques sont de la forme e, $\{a_1, \ldots, a_n\}$, $\emptyset$ ou Ω. Étudions les propriétés de la réécriture d'un ensemble E dans le λ-terme noté E' :

(a) $E = e \longrightarrow E' = \lambda x.\, E(x)$ avec $t(E) = 1$, $t^*(E') = 2$, $o(E') = 1$.

(b) $E = \{a_1, \ldots, a_n\} \longrightarrow E' = \lambda x.\, (x = a_1 \vee \cdots \vee x = a_n)$ avec $t(E) = t(a_1) + \cdots + t(a_n) + 1$, $t^*(E') = 3 * n + t(a_1) + \cdots + t(a_n)$, $o(E') = n$.

(c) $E = \emptyset \longrightarrow E' = \lambda x.\, \bot$ avec $t(E) = 1$, $t^*(E') = 1$, $o(E') = 0$.

(d) $E = \Omega \longrightarrow E' = \lambda x.\, \top$ avec $t(E) = 1$, $t^*(E') = 1$, $o(E') = 0$.

Notons que la formule augmente d'une taille maximum dans le cas $\{a_1, \ldots, a_n\}$ car dans ce cas $t(E') - t^*(E) = 3 * n - 1$. (1)

Les ensembles non-atomiques sont de la forme $E\, \alpha\, F$ (où α peut être $\cap, \cup$ ou $\setminus$). Comme dans la preuve du théorème 3.5, la réécriture puis la β-réduction de E et F terminent sur des λ-termes de la forme $\lambda y.\, E(y)$ et $\lambda z.\, F(z)$ avec E et F ne contenant pas de λ-termes (on note $\succcurlyeq_{\beta}^{\rightarrow}$ cette étape). Examinons les propriétés de la formule obtenue après réécriture et β-réduction de $E \cap F$:

$$
\begin{aligned}
E \cap F \longrightarrow\ & \big(\lambda PQ.\, \lambda x.\, P(x) \wedge Q(x)\big)\ (E, F) \\
& \succcurlyeq_{\beta}^{\rightarrow} \big(\lambda PQ.\, \lambda x.\, P(x) \wedge Q(x)\big)\ (\lambda y.\, E(y), \lambda z.\, F(z)) \\
& \succcurlyeq_{\beta} \lambda x.\, (\lambda y.\, E(y))(x) \wedge\ (\lambda z.\, F(z))(x)
\end{aligned}
$$

ici la β-réduction de $(\lambda y.\, E(y))(x)$ (respec. $(\lambda z.\, F(z))(x)$) aboutit à $E(x)$ qui est de taille $t^*(E)$ (respec. $t^*(F)$).

$$
\begin{aligned}
& \succcurlyeq_{\beta} E' = \lambda x.\, E(x) \wedge F(x) \\
& \text{avec } t(E \cap F) = t(E) + t(F) + 1 \\
& \qquad t(E') = t^*(E) + t^*(F) + 3 \\
& \qquad \text{et } o(E') = o(E) + o(F).
\end{aligned}
$$

De même :

$$
\begin{aligned}
E \cup F \longrightarrow E' \text{ où } & t(E \cup F) = t(E) + t(F) + 1 \\
& t(E') = t^*(E) + t^*(F) + 3 \\
& \text{et } o(E') = o(E) + o(F).
\end{aligned}
$$

et

$$
\begin{aligned}
E \setminus F \longrightarrow E' \text{ où } & t(E \setminus F) = t(E) + t(F) + 1 \\
& t(E') = t^*(E) + t^*(F) + 4 \\
& \text{et } o(E') = o(E) + o(F). \quad (2)
\end{aligned}
$$

Dans les trois cas, on a $o(E')$ qui a la même valeur. Ainsi pour obtenir une formule la plus grande possible, il faut avoir une expression initiale de la forme $E \setminus F$ (car dans ce cas on maximise $t(E') - t(E\, \alpha\, F) = t^*(E) + t^*(F) + 4$ (où α est $\cup, \cap$ ou $\setminus$)). Grâce à (1), on sait de plus que pour maximiser $t(E') - t(E\, \alpha\, F)$, il faut que E et F soient des énumérations (car il faut maximiser $t(E')$ qui est égal à $t^*(E) + t^*(F)$). (3)

Sans perte de généralité supposons que E soit de la forme $\{a_1, \ldots, a_n\}$ et F soit $\{b_1, \ldots, b_m\}$ (où $a_1, \ldots, a_n, b_1, \ldots, b_m$ sont des termes). Notons E' le λ-terme obtenu après transformation de $E \setminus F$, on a :

$$t(E \setminus F) = t(E) + t(F) + 1$$
$$= t(a_1) + \cdots + t(a_n) + 1 + t(b_1) + \cdots + t(b_m) + 1 + 1$$

Or, on a également :

$$t(E') = t^*(E) + t^*(F) + 4$$
$$= 3 * n + t(a_1) + \cdots + t(a_n) + 3 * m + t(b_1) + \cdots + t(b_m) + 4$$

donc $t(E') - t(E \setminus F) = 3 * n + 3 * m + 1$ et $o(E') = m + n$. Dans ce cas on maximise $o(E')$ et $t(E')$.

On sait que la transformation d'un ensemble E termine sur un terme de la forme $\lambda x.E'(x)$. C'est pourquoi il faut connaître le nombre de fois que x apparaît dans $E'(x)$ (il apparaît un nombre $o(\lambda x.E'(x))$ de fois) car cela influence la taille de la formule après transformation. En effet, la transformation de la formule $v \in E$ donne :

$$v \in E \longrightarrow (\lambda xP.\, P(x))(v, \lambda x.E'(x))$$

> Quand on β-réduit x, on remplace chacune de ses occurences par v. C'est pourquoi v peut-être dupliqué.
>
> $\succcurlyeq_\beta E'(v)$, dans cette formule v apparaît un nombre de fois égal à $o(\lambda x.\, E'(x))$.

Si E est une énumération (et c'est le cas qui nous intéresse car c'est celui qui génère la plus grosse formule comme montré en (3)), la formule obtenue $E'(v)$ a une taille égale à $o(E) * (t(v) + 2)$. (4)

Soient E et F des ensembles. Les formules booléennes ensemblistes possibles sont $v \in E$, $E \subseteq F$, $E \subset F$ et $E = F$. Nous pensons que la transformation de $v \in E$ est celle où on va avoir la plus grande formule car le terme v va être dupliqué (pour démontrer la conjecture il faudrait étudier comment la grandeur $o(.)$ se comporte sur les différents ensembles possibles).

Supposons que v soit de taille k. On a vu en (3) que le cas où la transformation d'un ensemble E' termine sur la plus grosse formule est celui où E' est de la forme $E \setminus F$ avec E et F des énumérations (par exemple $\{a_1, \ldots, a_n\}$ et $\{b_1, \ldots, b_m\}$ (où $a_1, \ldots, a_n, b_1, \ldots, b_m$ sont des termes)). On a alors $t(v \in E') = t(a_1) + \cdots + t(a_n) + t(b_1) + \cdots + t(b_m) + k + 1$. De plus grâce à (4), on sait que la transformation de $v \in E'$ finit sur une formule de taille égale à $(n + m) * (k + 2)$ (car grâce à (2) on sait que v apparaît $o(E') = n + m$ fois).

Prenons $t(a_1) + \cdots + t(a_n) + t(b_1) + \cdots + t(b_n) = n + m = n/2$ et $k = n/2$. Dans ce cas, a partir d'une formule de taille $n+3$, on obtient, après transformation, une formule de taille $(n/2 + n/2) * (n/2 + 2) = n * (n/2 + 2)$. Cette valeur est équivalente à n^2, la réécriture puis la β-réduction produit donc une formule de taille quadratique par rapport à la taille de la formule d'entrée.

En démontrant un comportement identique pour $E \subseteq F$, $E \subset F$ et $E = F$, on pourrait conclure. En effet, comme l'application de l'extensionnalité sur une formule de taille n produit une formule de taille $O(n)$, nous pourrions affirmer que la transformation d'une formule SET de taille n en formule FOL produit un arbre syntaxique de taille équivalente à n^2 (comportement quadratique).

B Exemples d'utilisation

Voici un exemple d'interaction de l'utilisateur avec Isabelle/HOL (en utilisant l'interface générique d'Isabelle : Proof General, voir [Asp00]) et les tactiques que j'ai écrites. La commande **apply** permet d'appliquer une tactique et la commande **by** permet de finir une preuve avec la tactique qui suit. A chaque fois, on voit le résultat de la traduction de SET vers FOL comme nouveau sous-but. Ensuite la tactique *rv_fol* qui appelle haRVey pour des formules FOL conclut la preuve. A noter que le deuxième exemple utilise la méthode de réflexion car les sous-formules ensembliste ne contiennent que des ensembles explicites. Le quatrième exemple montre qu'on peut utiliser l'axiome de compréhension (qui permet de définir un ensemble à partir d'un prédicat) :

theory exemples
imports harvey
begin

lemma "$((X \cap Y) = \emptyset) \longrightarrow ((X \setminus Y) = X)$"
  apply set2fol
  1 subgoal :
    1. "$(\forall x.(P\ x\ \wedge\ Q\ x) = \bot) \longrightarrow (\forall x.(P\ x\ \wedge\ \neg Q\ x) = P\ x)$"
  by rv_fol

lemma "$((\emptyset \subseteq \{a, b\})\ \vee\ (\{a\} \subseteq \{a, b, c\}))\ \wedge\ (a \neq b\ \vee\ f\ a = b\ \vee\ g\ b \neq f\ a\ \vee\ ((\exists x.b \neq g\ x)))$
  $\wedge\ (P\ x\ \vee\ \neg P\ x\ \vee\ Q\ x\ \vee\ (R\ y \longrightarrow S))$"
  apply set2fol_reflex
  1 subgoal :
    1. "$((\forall x.\ \bot \longrightarrow x = a\ \vee\ x = b \vee\ \bot)$
    $\vee\ (\forall x.x = a \vee\ \bot \longrightarrow x = a\ \vee\ x = b\ \vee\ x = c \vee\ \bot))$
    $\wedge\ (a \neq b\ \vee\ f\ a = b\ \vee\ g\ b \neq f\ a\ \vee\ (\exists x.b \neq g\ x))\ \wedge\ (P\ x\ \vee\ \neg P\ x\ \vee\ Q\ x\ \vee\ (R\ y \longrightarrow S))$"
  by rv_fol

lemma "$(D \cap B) \cup (C \cap D) \cup (B \cap C) = (B \cup C) \cap (C \cup D) \cap (D \cup B)$"
  apply set2fol
  1 subgoal :
    1. "$\forall x.((Q\ x \wedge Px \vee R\ x \wedge Q\ x) \vee Px \wedge R\ x) =$
    $(((Px \vee R\ x) \wedge (R\ x \vee Q\ x)) \wedge (Q\ x \vee Px))$"
  by rv_fol

lemma "$(\forall x.P\ x = (\neg I\ x)) \longrightarrow (\{x.P\ x\} \cap \{x.I\ x\} = \emptyset)$"
  apply set2fol
  1 subgoal :
    1. "$(\forall x.P\ x = (\neg I\ x)) \longrightarrow (\forall x.(P\ x \wedge I\ x) = \bot)$"
  by rv_fol

end

Références

[Asp00] D. Aspinall, « Proof general : A generic tool for proof development », dans *TACAS '00 : Proceedings of the 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, p. 38–42, London, UK, 2000. Springer-Verlag.

[Cha06] A. Chaieb, « Mechanized quantifier elimination for linear real-arithmetic in Isabelle/HOL. ». Rapport technique, Technische Universität München, 2006.

[FMM+06] P. Fontaine, J.-Y. Marion, S. Merz, L. P. Nieto et A. Tiu, « Expressiveness + automation + soundness : Towards combining SMT solvers and interactive proof assistants », dans H. Hermanns et J. Palsberg, éditeurs, *12th Intl. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2006)*, vol. 3920 (coll. *Lecture Notes in Computer Science*), p. 167–181, Vienna, Austria, 2006. Springer-Verlag.

[Fon04] P. Fontaine, *Techniques for verification of concurrent systems with invariants*. Thèse de doctorat, Institut Montefiore, Université de Liège, Belgium, septembre 2004.

[GL01] J. Goubault-Larrecq, *Logique classique du premier ordre*. 2001.

[Har95] J. Harrison. « Metatheory and reflection in theorem proving : A survey and critique », 1995.

[HRCS02] J. Hooker, G. Rago, V. Chandru et A. Shrivastava. « Partial instantiation methods for inference in first order logic », 2002.

[Hur99] J. Hurd, « Integrating gandalf and HOL », dans *Theorem Proving in Higher Order Logics*, p. 311–322, 1999.

[MQPss] J. Meng, C. Quigley et L. C. Paulson, « Automation for interactive proof : First prototype », *Information and Computation*, in press.

[NBH00] H. de Nivelle, M. Bezem et D. Hendriks, « Automated proof construction in type theory using resolution », dans D. McAllester, éditeur, *Proceedings of the 17th International Conference on Automated Deduction (CADE-17)*, vol. 1831 (coll. *Lecture Notes in Artificial Intelligence*), p. 148–163, Carnegie Mellon University, Pittsburgh, PA, USA, June 2000. Springer.

[NBS06] T. Nipkow, G. Bauer et P. Schultz, « Flyspeck I : Tame graphs », dans U. Furbach et N. Shankar, éditeurs, *Automated Reasoning (IJCAR 2006)*, 2006. To appear.

[Non96] A. Nonnengart. « Strong skolemization », 1996.

[NPW02] T. Nipkow, L. Paulson et M. Wenzel, *Isabelle/HOL. A Proof Assistant for Higher-Order Logic*. Springer-Verlag, 2002.

[ORS92] S. Owre, J. M. Rushby, et N. Shankar, « PVS : A prototype verification system », dans D. Kapur, éditeur, *11th International Conference on Automated Deduction (CADE)*, vol. 607 (coll. *Lecture Notes in Artificial Intelligence*), p. 748–752, Saratoga, NY, jun 1992. Springer-Verlag.

[RT05] S. Ranise et C. Tinelli, « The smt-lib standard : Version 1.1 ». Rapport technique, 2005.

[The05] The Coq development team, « The coq proof assistant reference manual v8.0 ». Rapport technique n° 255, INRIA, France, mars 2005.

[Web05] T. Weber, « Integrating a SAT solver with an LCF-style theorem prover », dans A. Armando et A. Cimatti, éditeurs, *Proceedings of PDPAR'05 – Third International Workshop on Pragmatical Aspects of Decision Procedures in Automated Reasoning*, Edinburgh, UK, juillet 2005.