Recherche de preuve efficace dans une logique à points fixes

Alexandre Viel, Dale Miller, PARSIFAL

septembre 2008

Le contexte général

Dans la théorie des types et des langages de programmations, des systèmes de complexité croissante sont étudiés. Le raisonnement sur ces systèmes devient ainsi plus long et fastidieux, prouver des résultats importants comme la correction du typage d'un langage de programmation ou la normalisation forte d'un calcul demande de faire des preuves de plus en plus longues alors que le principe derrière ces preuves reste quasi inchangé. Aussi, pouvoir raisonner de manière automatique sur des spécifications logiques et automatiser une partie de ces preuves devient de plus en plus intéressant et important.

Le problème étudié

L'équipe de Miller a développé récemment une logique disposant de bons outils pour décrire des spécifications logiques, qui réunit en particulier la quantification générique nécessaire pour spécifier les liaisons de variables, avec des opérateurs de point fixes permettant de raisonner sur des prédicats inductifs ou coinductifs. Mon travail a été de rendre la recherche de preuve dans cette logique viable, capable de montrer les résultats dont les preuves suivent un schéma d'induction simple et dont une grosse partie n'a rien d'intéressant ou de difficile. Des questions se lèvent alors naturellement : Comment intégrer la preuve par induction dans une tactique focalisée de recherche de preuve ? Comment, tout en restant dans un contexte très général, déterminer l'hypothèse d'induction ou l'invariant qui fait fonctionner la preuve ? Comment décider du bon moment pour tenter une induction ?

La contribution proposée

Une étude de la polarité des règles et du comportement des règles spécifiques à l'induction avec les autres règles permet de limiter les cas d'applications de ces règles afin de laisser le moins de choix équivalents possible pour le prouveur. On peut aussi différencier les formules qui représentent des calculs des formules véritablement logiques. Cela permet dans certains cas au prouveur d'acquérir un comportement un peu similaire à un système plus destiné à être calculatoire comme un système de réécriture.

Quant à l'inférence d'un invariant, une inférence complète n'est pas envisageable, mais il est cependant possible de procéder à une inférence partielle si on accepte de perturber légèrement le comportement focalisé de la recherche de preuve et de préciser le problème plus général de l'instanciation de variables en recherche de preuve.

Les arguments en faveur de sa validité

Pouvoir relier l'effet des techniques obtenues au comportement d'autres systèmes montre que ces techniques sont intéressantes et permet de les utiliser tout en restant dans un contexte très général. En implémentant une tactique focalisée de recherche de preuve, on obtient des résultats satisfaisants, et on montre comment les problèmes résolus peuvent donner une tactique plus puissante.

Le bilan et les perspectives

Dans le cadre d'analyse de programmes, on dispose déjà de raisonnements comme l'inférence de type et l'interprétation abstraite qui devraient pouvoir être simulés par de la recherche de preuve avec induction plus ou moins fidèlement. Il est toujours envisageable d'en tirer des techniques spécifiques qui peuvent être alors étendues à la recherce de preuve en général : l'étude de la transcription de ces méthodes peut mettre en évidence des principes utiles à combiner aux méthodes déjà existantes pour continuer de renforcer la recherche de preuve. D'autre part, les techniques proposées soulèvent encore d'autres questions sur le comportement d'un prouveur face à l'inférence d'invariants ou de métavariables, ainsi que la manière dont un prouveur peut utiliser les résultats qu'il prouve pour modifier la recherche de preuve elle-même.

Table des matières

1 la logique $\mu MALL^{=}$ **4**
1.1 Définition 4
1.2 Propriétés des points fixes 5
1.3 Le prouveur interactif **Taci** 7

2 L'inférence d'un invariant **8**
2.1 Le contexte comme invariant 8
2.2 Exemple : nat 9
2.3 L'implémentation dans **Taci** 10
2.4 Induction forte 10

3 Raffinement de la génération d'invariant **12**
3.1 L'affaiblissement 12
3.2 Métavariable ou métaformule 13
3.3 Autres raffinements 14

4 Les types de données **15**
4.1 La notion de progrès 15
4.2 La structure des matchers 17
4.3 La Skolémisation 19
4.3.1 Exemple 20

5 Conclusion et travaux futurs **22**

Mon travail a été de justifier et de mettre en place dans le prouveur taci une tactique de recherche de preuve prenant en compte la focalisation de la logique et capable de réaliser des inductions pour montrer des résultats simples sur des spécifications logiques de manière purement automatique.

Le prouveur NqThm de Boyer et Moore, [4] a montré par exemple qu'une bonne classe de résultats d'arithmétique pouvaient être montrés à l'aide d'un prouveur doté de bonnes heuristiques.

La logique utilisée est intéressante car elle permet de raisonner effectivement sur les spécifications logiques [10]. On peut donc espérer pouvoir en tirer un prouveur capable de traiter automatiquement une partie des preuves portant sur la méta-théorie des systèmes logiques ou des langages de programmation.

On cherche aussi à voir si cette logique pourrait simuler divers procédés d'analyse statique comme le typage, ou des procédures plus spécialisées (par exemple des procédures utilisées dans le système Twelf pour montrer des propriétés sur les fonctions définies par l'utilisateur [1], [8]), et ainsi construire un système qui pourrait être utilisé pour étudier des programmes et en déduire rapidement non seulement leur type, mais peut-être d'autres spécifications simples demandées par l'utilisateur.

1 la logique $\mu MALL^{=}$

1.1 Définition

La logique $\mu MALL^{=}$ présentée par David Baelde dans [6] est une extension de la logique linéaire utilisant des constructeurs de plus grand et plus petit point fixe.

Les formules sont construites à partir des connecteurs de la logique linéaire $\otimes$,$\oplus$,$\parr$,&,$\exists$,$\forall$. Les règles associées sont les règles usuelles :

$$\frac{\vdash \Gamma, P \qquad \vdash \Delta, Q}{\vdash \Gamma, \Delta, P \otimes Q} \otimes \quad \frac{\vdash \Gamma, P, Q}{\vdash \Gamma, P \parr Q} \parr$$

$$\frac{\vdash \Gamma, P \qquad \vdash \Gamma, Q}{\vdash \Gamma, P \,\&\, Q} \,\& \quad \frac{\vdash \Gamma, P_1}{\vdash \Gamma, P_1 \oplus P_2} \oplus_1 \quad \frac{\vdash \Gamma, P_2}{\vdash \Gamma, P_1 \oplus P_2} \oplus_2$$

$$\frac{\vdash \Gamma, P \qquad x \text{ libre dans } \Gamma}{\vdash \Gamma, \forall x P} \forall \quad \frac{\vdash \Gamma, P[x \leftarrow t]}{\vdash \Gamma, \exists x P} \exists$$

La structure du 1er ordre est donnée par les règles de l'égalité :

$$\frac{}{\vdash t = t} \qquad \frac{\{\vdash \Gamma\theta, \theta \in csu(u, v)\}}{\vdash \Gamma, u \neq v} \neq$$

Ici, *csu* désigne un ensemble complet d'unificateurs.
Pour simplifier, je supposerai que les termes sont des termes du 1er ordre d'un seul type γ. L'ensemble d'unificateurs est alors soit vide dans le cas où l'unification est impossible, soit un singleton comprenant l'unificateur le plus général. L'égalité et l'inégalité sont duales l'une de l'autre.

La négation est définie par dualité : Les règles d'initialisation et de coupures sont :

$$\frac{}{\vdash P^\perp, P}\ init \qquad \frac{\vdash \Gamma, P^\perp \qquad \vdash P, \Delta}{\vdash \Gamma, \Delta}\ cut$$

La logique a les constructeurs de plus petit et plus grands points fixes μ et ν, qui sont le dual l'un de l'autre. Le corps B d'un point fixe μ_n ou ν_n opère sur les prédicats d'arité n, de type $\tau_n = \gamma \to \cdots \gamma \to o$. B est de type $\tau_n \to \tau_n$, et μ_n et ν_n sont de type $(\tau_n \to \tau_n) \to \tau_n$.

L'arité du point fixe pouvant être lue dans le corps du point fixe, on peut généralement se passer de la préciser. Le dual d'un plus petit point fixe μBt est le plus grand point fixe $\nu B^\perp t$ où on définit $B^\perp$ par

$$B^\perp := \lambda p.\lambda x.\left(B(p^\perp)x\right)^\perp$$

et $p^\perp = \lambda x.(px)^\perp$.

Les règles d'inférence des points fixes sont

$$\frac{\vdash S^\perp x, BSx \qquad \vdash St, \Gamma}{\vdash \nu Bt, \Gamma}\ \nu \qquad \frac{\vdash B(\mu B)t, \Gamma}{\vdash \mu Bt, \Gamma}\ \mu$$

L'opération de dualisation ne faisant pas partie de la logique, les points fixes considérés sont tous monotones : les occurences récursives du prédicat p dans Bp sont positives car elles ne peuvent pas être niées.
Dans la règle d'induction (ν), S est un prédicat de même arité que B. Lorsque la prémisse de gauche est prouvable, on dit que le prédicat S est un invariant de B

Cette logique admet l'élimination des coupures ainsi qu'un théorème de focalisation.

On peut définir l'implication linéaire $P \to Q$ comme sucre syntaxique pour $P^\perp \parr Q$, et les séquents à deux cotés $\Gamma \vdash \Delta$ comme $\vdash \Gamma^\perp, \Delta$.

1.2 Propriétés des points fixes

Pour discuter d'une éventuelle polarité des points fixes et du fonctionnement de ces deux règles pendant une recherche de preuve, il faut observer quelques propriétés de base.

Tout d'abord, la propriété de monotonicité d'un point fixe permet d'admettre une règle importante :
Si x n'apparaît pas dans les prédicats P_1 et P_2, la règle suivante est admissible :

$$\frac{\vdash P_1 x, P_2 x}{\vdash B^\perp P_1 t, BP_2 t}\ mon$$

En effet, en faisant jouer la dualité de B et $B^\perp$ on peut décomposer B jusqu'à arriver aux occurences récursives de B et n'obtenir que des prémisses de la forme $\vdash P_1 x, P_2 x$.
La règle μ permet seulement d'effectuer un déroulage du plus petit point fixe.
La règle ν est la règle d'induction, elle remplace le plus grand point fixe par n'importe quel autre point fixe S de B. La première prémisse est la preuve de l'invariance de S par B, et apparaît sans rapport avec le reste de la preuve puisqu'elle ne fait intervenir que B et S, le

contexte n'apparaît que dans la seconde prémisse.
La deuxième règle utile à admettre est la règle de déroulage du plus grand point fixe : L'équivalent de la règle μ est facile à dériver en choisissant $B(\nu B)$ comme invariant

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \mu B^{\perp} x, \nu B x}\, init}{\vdash B^{\perp}\mu B^{\perp} x, B\nu B x}\, mon}{\vdash \mu B^{\perp} x, B\nu B x}\, \mu}{\vdash B^{\perp}\mu B^{\perp} x, B(B\nu B)x}\, mon \qquad \vdash B\nu B t, \Gamma}{\vdash \nu B t, \Gamma}\, \nu$$

Un autre exemple est la possibilité de remplacer un ν par un μ, en choisissant cette fois μB comme invariant (ce qui montre que μB est bien un point fixe de B)

$$\dfrac{\dfrac{\dfrac{}{\vdash B^{\perp}\nu B^{\perp} x, B\mu B x}\, init}{\vdash \nu B^{\perp} x, B\mu B x}\, \nu_2 \qquad \vdash \mu B t, \Gamma}{\vdash \nu B t, \Gamma}\, \nu$$

On peut de même montrer que νB est aussi un invariant. Ensuite on peut montrer que $\mu B t$ et $B(\mu B)t$ sont équivalents, ainsi que $\nu B t$ et $B(\nu B)t$, et que le plus petit point fixe implique le plus grand, ce qui justifie les noms donnés à μ et à ν.

Du point de vue commutativité et polarité, les règles (μ) et (ν) ne font que remplacer un point fixe par une formule plus compliquée et n'agissent pas sur le reste du contexte. Ainsi, ces deux règles commutent aisément avec toutes les règles qui ne dupliquent pas leur contexte. La seule règle qui peut poser problème est la règle (&) puisqu'elle crée deux prémisses, chacune gardant le même contexte.

Si un plus petit point fixe μ est dupliqué, il ne peut lui arriver que deux choses, un déroulage, ou une règle d'axiome. Comme on a montré que la règle de déroulage était admissible pour ν, il est possible de faire commuter la règle (μ) devant (&) même s'il n'était pas destiné à être déroulé dans l'autre branche de la dérivation.

Si un plus grand point fixe ν est dupliqué, il peut être remplacé par deux invariants différents (dans le cas où on utilise la règle d'axiome sans utilise la règle d'induction dans la seconde branche, on peut toujours rajouter une règle d'induction transparente en utilisant νB comme invariant de B). La commutation de la règle (ν) avec (&) repose alors sur le fait que si S_1 et S_2 sont deux invariants de B, alors $S_1 \oplus S_2$ en est toujours un.

L'autre règle qui pose problème pour (ν) est la règle ($\forall$), qui introduit une variable y. De la même manière que pour (&), l'invariant peut dépendre de cette variable y. La solution est la même : si $\lambda x.Syx$ est un invariant de B, alors $\lambda x.\exists y, Syx$ est encore un invariant.
Cette propriété permet de demander à ce que les invariants n'aient pas de variable libre, puisqu'on peut toujours les quantifier existentiellement. On supposera que ce sera le cas dans la suite.

Au vu de ces règles, un point fixe n'a a priori pas beaucoup d'interaction avec le reste de la formule, et on peut décider à n'importe quel moment de ce que va devenir un point fixe.

Pour un μ, s'il doit être déroulé ou s'il sera utilisé dans une règle d'axiome ; pour un ν, savoir par quel invariant il va être remplacé ou s'il sera utilisé dans une règle d'axiome.

Si on devait donner une polarité à ces deux connecteurs, on devrait pouvoir choisir entre μ et ν lequel est asynchrone et lequel est synchrone. Cependant, lorsque B est totalement asynchrone ou totalement synchrone, choisir μB de la même polarité que B fait apparaître le comportement infini du point fixe : on risque d'entrer dans une phase synchrone qui présente une infinité de choix possibles si B est synchrone, ou dans une phase asynchrone infinie si B est asynchrone

C'est pourquoi on est plutôt amené à classer μ comme synchrone et ν comme asynchrone. Mais a priori, comme les deux règles commutent avec le reste, l'autre option reste envisageable.

Cependant, en recherche de preuve dirigée par le but, les preuves prennent une autre tournure.

1.3 Le prouveur interactif Taci

Taci est un prouveur interactif développé par l'équipe PARSIFALdepuis l'été 2007, et disponible en ligne depuis cet été [9] Le principe de **Taci** est de pouvoir implémenter et tester rapidement différentes tactiques de preuves automatiques sur différentes logiques. Chaque logique est implémentée dans son propre module, voire foncteur si la logique dépend de paramètres, où sont définies les règles d'inférences de la logique et éventuellement des stratégies de recherche de preuve basées sur des résultats de focalisation [2].

J'ai travaillé sur une logique du permier ordre avec point fixe et quantification générique. Plutôt que d'avoir quatre connecteurs $\otimes$ $\oplus$ & $\parr$ au comportement logique et aux polarités différentes, on utilise deux connecteurs classiques $\wedge$ et $\vee$, qu'on décline en $\wedge^+$ $\wedge^-$ $\vee^+$ $\vee^-$. Les connecteurs étant alors considérés comme synchrones et les négatifs comme asynchrones. En indiquant quelle polarité donner aux connecteurs, l'utilisateur peut agir sur le comportement du prouveur lorsqu'il utilise une stratégie focalisée.

La tactique de preuve, sans points fixes, est une recherche de preuve focalisée :

Pendant une phase asynchrone, le prouveur applique toutes les règles asynchrones possibles aux formules du séquent manipulé, sans se préoccuper des choix qu'il est en train de faire. A la fin d'une phase asynchrone, toutes les formules présentes dans le séquent sont synchrones. C'est à partir de ce moment là que le prouveur doit se souvenir de ses choix pour avoir la possibilité de faire un retour en arrière si la preuve échoue.

Le prouveur sélectionne et se focalise sur une formule du séquent, et applique toutes les règles synchrones possibles sur cette formule, en se donnant à chaque choix qu'il est amené à faire, la possibilité d'un retour en arrière en cas d'échec de la recherche.

La preuve d'une formule $\exists x.Px$ s'effectue en introduisant une métavariable X à la place de x. Dans le reste de la recherche, cette métavariable est partiellement ou totalement unfiée avec les termes présents dans le séquent au moment où cette métavariable est introduite. Ce processus d'unification ne fonctionne pas toujours et est particulièrement sensible à l'ordre dans lequel le prouveur choisit les séquents qu'il essaye de prouver.

Lorsqu'une preuve échoue, le prouveur backtracke alors au dernier point de choix effectué pendant une phase synchrone. L'intérêt de la focalisation est de limiter les points de choix à

la phase synchrone et de rendre les phases asynchrones totalement transparentes vis-à-vis du backtracking.

Une partie de mon travail a donc été d'intégrer le principe d'induction et la génération d'invariant à cette tactique déjà en place.

2 L'inférence d'un invariant

Un prouveur automatique ne peut pas connaître à l'avance l'invariant qui va convenir à la preuve. Une recherche qui se voudrait la plus générale possible devrait introduire un méta-invariant, travailler d'abord dans la branche $\vdash St, \Gamma$, en découvrant la structure de S au fur et à mesure qu'on prouve Γ, puis ensuite travailler dans la preuve de l'invariance de l'invariant partiellement inferré.
Cette fois, on a donc un ordre implicite sur le traitement des prémisses, et on a une interaction entre la preuve de la prémisse de droite vers la prémisse de gauche. Le contexte n'est plus totalement étranger à l'invariant. On pourrait d'ailleurs séparer la règle d'induction (ν) du choix de l'invariant S en décomposant la règle avec une coupure qui introduit l'invariant utilisé :

$$\frac{\vdash S^{\perp}x, BSx \qquad \vdash St, \Gamma}{\vdash \nu Bt, \Gamma}\,\nu \qquad \frac{\dfrac{\vdash BSx, S^{\perp}x}{\vdash \nu Bt, S^{\perp}t}\,\nu' \qquad \vdash St, \Gamma}{\vdash \nu Bt, \Gamma}\,cut$$

La règle d'induction ne serait alors rien d'autre qu'une coupure combinée avec une règle d'induction faible (ν') plus simple. Bien sûr, si on remplaçait la règle d'induction par celle ci, cette coupure deviendrait impossible à éliminer. Même si elle ne remet pas en cause la cohérence de la logique, il s'agit donc d'une coupure nécessaire à son bon fonctionnement.
Le premier choix raisonnable possible pour un prouveur automatique est donc de rechercher une preuve utilisant cette règle d'induction faible mais ne cherchant cependant pas à inventer une coupure à chaque induction.

2.1 Le contexte comme invariant

Pour montrer $\vdash \nu Bt, \Gamma$, on va donc choisir S de manière à obtenir $\vdash \Gamma^{\perp}, \Gamma$ dans la prémisse de droite après réduction. Notons Σ l'ensemble des variables libres de Γ. t et Γ dépendent alors de Σ, on peut les noter t^{Σ} et Γ^{Σ} si il y a plusieurs environnement disponibles afin d'éviter les confusions.
Notre invariant doit recopier l'environnement Σ en un environnement Σ' :

$$S := \lambda x.\exists\Sigma.x = t^{\Sigma} \otimes (\Gamma^{\perp})^{\Sigma}$$

La rège d'induction devient alors :

$$\frac{\dfrac{\dfrac{\dfrac{\Sigma' \vdash BSt^{\Sigma'}, \Gamma^{\Sigma'}}{\Sigma', x \vdash BSx, x \neq t^{\Sigma'}, \Gamma^{\Sigma'}}\,\neq}{\Sigma', x \vdash BSx, x \neq t^{\Sigma'} \parr \Gamma^{\Sigma'}}\,\parr}{x \vdash BSx, S^{\perp}x}\,\forall \qquad \frac{\dfrac{\dfrac{}{\Sigma \vdash \Gamma^{\perp}, \Gamma}\,init \quad \dfrac{}{\Sigma \vdash t^{\Sigma} = t^{\Sigma}}}{\Sigma \vdash t^{\Sigma} = t^{\Sigma} \otimes (\Gamma^{\perp})^{\Sigma}, \Gamma^{\Sigma}}\,\otimes}{\Sigma \vdash \exists\Sigma'.t^{\Sigma} = t^{\Sigma'} \otimes (\Gamma^{\perp})^{\Sigma'}, \Gamma^{\Sigma}}\,\exists}{\Sigma \vdash \nu Bt^{\Sigma}, \Gamma^{\Sigma}}\,\nu$$

Soit :

$$\frac{\vdash BSt, \Gamma}{\vdash \nu Bt, \Gamma}$$

où S est un prédicat qui correspond à l'hypothèse d'induction, construit directement sur $\Gamma^\perp$
Alors qu'auparavant on ne pouvait pas rapprocher la preuve de l'invariant à la preuve du séquent de départ, on peut maintenant discuter de la dynamique de cette nouvelle règle.
Cette règle commute toujours avec les connecteurs asynchrones $\forall$ ⅋ $\neq$ et partiellement avec & mais plus avec les connecteurs synchrones $\exists$ $\oplus$ $\otimes$, ce qui la classe cette fois-ci comme synchrone. Les modifications du séquent lors d'une utilisation de $\forall$ ⅋ ou $\neq$ n'ont aucun effet sur l'invariant, tandis que utiliser une règle (&) avant deux inductions peut être remplacé par une seule induction puis une utilisation de (&).
Bien sûr, dès qu'on restreint la forme de l'invariant, on perd la complétude vis-à-vis du système initial, En ajoutant en plus la règle de déroulage du ν, on obtient déjà une procédure qui peut montrer quelques résultats simples.

2.2 Exemple : nat

On définit les entiers naturels par

$$Nat := \lambda nat.\lambda x.x = 0 \oplus (\exists y.x = sy \otimes nat\, y)$$

ainsi que les entiers pairs par

$$Even := \lambda even.\lambda x.x = 0 \oplus (\exists y.x = s(sy) \otimes even\, y)$$

le plus petit point fixe μNat définit le prédicat être un entier.
Si on recherche une preuve de

$$\vdash \forall x.(\mu Natx)^\perp, Px$$

En supposant que P est un prédicat synchrone ou un point fixe, une utilisation de la règle d'induction précédente choisit P comme invariant, et ressemble à :

$$\dfrac{\dfrac{\dfrac{\vdash P\,0 \qquad \dfrac{\vdash (P\,x)^\perp, P\,(s\,x)}{\vdash \exists x'.y = x' \otimes (P\,x')^\perp, P(s\,y)}\,sync}{\vdash Nat^\perp P\,x, P\,x}\,async}{\vdash (\mu Nat\,x)^\perp, P\,x}\,induction}{\vdash \forall x.(\mu Nat\,x)^\perp, P\,x}\,async$$

Une preuve de $\vdash \forall x.\mu Even\,x \to \mu Nat\,x$ se fait par induction sur $Even$:
Prouver que Nat est un invariant de $Even$ revient à prouver :

$$\vdash \mu Nat\,0 \qquad \vdash \mu Nat\,x \to \mu Nat\,s(s\,x)$$

Les deux prémisses se prouvent alors en une phase synchrone en déroulant μNat une fois à gauche et deux fois à droite.

2.3 L'implémentation dans Taci

La tactique de preuve développée combine les points fixes avec la focalisation. Pour ne pas rechercher de preuve trop longtemps, l'utilisateur doit donner une borne du nombre d'inductions imbriquées ou de déroulages synchrones nécessaires à la preuve.

Afin d'éviter d'avoir de trop nombreux choix même pendant une recherche focalisée, le prouveur décide de ce qu'il va faire d'un point fixe ν à la fin de la phase asynchrone où ce point fixe apparaît au niveau du séquent. On estime que si on doit faire une induction, on doit la faire immédiatement, c'est vrai quand on connaît à l'avance le bon invariant, mais on y perd lorsque l'on veut inférrer l'invariant à partir du contexte. Néanmoins, cette décision réduit fortement le nombre de choix du prouveur. Le but est de trouver les preuves simples et d'échouer rapidement sinon, on privilégie la rapidité à la complétude.

Aussi, à la fin d'une phase asynchrone, le prouveur explore pour chaque point fixe ν présent dans le séquent, les trois possibilités :

- geler le point fixe
- faire une induction
- faire un déroulage

Les deux derniers choix consument la borne d'induction donnée par l'utilisateur.

Lors d'une induction, dégeler les points fixes présents permet de ne pas avoir une tactique de recherche de preuve trop dépendante de l'ordre dans lequel les hypothèses sont présentes. L'autre alternative étant de retirer les points fixes gelés de l'invariant supposerait en effet que l'ordre dans lequel le prouveur examine les hypothèses est compatible avec l'ordre d'utilisation des points fixes.

2.4 Induction forte

L'une des manières d'avoir un invariant plus fort tout en restant général est d'utiliser l'induction forte. Ce principe n'est pas spécifique aux entiers naturels mais peut être appliqué à n'importe quel point fixe. Ainsi à chaque opérateur de point fixe B d'arité n on peut définir un opérateur ordre B_* d'arité $2n$:

$$B_* := \lambda P.\lambda x.\lambda y.x = y \oplus B(Px)y$$

A partir de l'ancien invariant S on construit :

$$S_* := \lambda x.\exists y.\mu B_* yx \otimes Sy$$

On peut alors dériver :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \mu B_*\, y\, z, \nu B_*^\perp\, y\, z} \qquad \dfrac{}{\vdash S\, y, S^\perp\, y}}{\vdash S_*\, z, \nu B_*^\perp\, y\, z, S^\perp\, y}}{\vdash BS_*\, x, S^\perp\, x \qquad \vdash BS_*\, x, B^\perp(\nu B_*^\perp\, y)\, x, S^\perp\, y}}{\vdash BS_*\, x, \nu B_*^\perp\, y\, x, S^\perp\, y}}{\vdash BS_*\, x, (S_*)^\perp\, x}$$

et

$$\dfrac{\dfrac{\dfrac{}{\vdash t = t}}{\vdash t = t \oplus B(\mu B_*\, t)\, t} \qquad \vdash S\, t, \Gamma}{\vdash \mu B_*\, t\, t \otimes S\, t, \Gamma}}{\vdash S^*\, t, \Gamma}$$

Ce qui permet d'avoir la règle

$$\dfrac{\vdash BS_*\, x, S^\perp\, x \qquad \vdash S\, t, \Gamma}{\vdash \nu B\, t, \Gamma}$$

Maintenant combinée avec le choix d'invariant précédent, on peut alors utiliser la règle

$$\dfrac{\vdash BS_*\, t, \Gamma}{\vdash \nu B\, t, \Gamma}$$

où S_* est construit à partir de B_* et Γ. C'est une version qui se veut plus forte de la règle précédente.

On peut utiliser νB_* comme μB_* dans la construction de S_* ; dans la dérivation de cette règle, seules les deux règles de déroulage sont utilisées (il n'y a pas d'induction sur νB_*). Choisir νB_* à la place de μB_* donnera plus d' opportunités d'appliquer l'hypothèse d'induction, mais au prix d'une induction à faire en plus (ce qui n'est pas tellement le but).

Dans le cas où B est $Nat^\perp$, on peut calculer B_* et S_*.
$B_*\, x\, y$ veut bien dire que $x \leqslant y$. Cette définition de l'ordre, construite sur le dual de Nat, détruit y jusqu'à arriver sur un sous-terme de y égal à x. Dans le cas des listes, on aurait x comme suffixe de y. Dans le cas des arbres, x serait un sous-arbre dans une branche de y.
L'ordre utilisé est intrinsèque au point fixe, on a pas d'autre choix possible. Les autres ordres qu'on pourrait vouloir définir (par exemple x préfixe de y, ou plus généralement y est une extension de x) ne sont pas généralisables aux points fixes quelconques.

Cette règle peut être sympathique dans un prouveur interactif, mais dans un prouveur automatique, l'hypothèse d'induction obtenue est plus difficilement utilisable. Auparavant, on savait à quel terme s'appliquait l'hypothèse d'induction. Ici, lorsque le prouveur veut l'utiliser il doit fournir un terme y dont il doit prouver qu'il est plus petit que x. Il utilise donc une métavariable à la place de y.
A partir de ce moment là, l'ordre dans lequel le prouveur continue sa recherche de preuve est crucial. Selon l'ordre dans lequel il décompose la conjonction $\mu B_* yx \otimes Sy$ de $S_* x$, et selon les polarisations présentes dans B et S, le succès pour une instanciation correcte de y est loin d'être garanti. Il faudrait utiliser un tenseur non commutatif ou une annotation qui pousse le prouveur à appliquer l'hypothèse d'induction avant de chercher à prouver qu'elle est applicable.

C'est un principe qu'on retrouve si on cherche à raffiner l'invariant : Il est plus important de chercher une utilisation de l'hypothèse d'induction qui fonctionne que de vérifier qu'on est dans des conditions où elle est applicable.

3 Raffinement de la génération d'invariant

Il n'est pas rare que le contexte disponible n'est pas un invariant, mais une légère modification du contexte permettrait de faire fonctionner la preuve d'invariance. Il faut alors se demander dans quelle mesure est-ce qu'on peut laisser une partie de l'invariant indéterminée sans trop gêner le prouveur dans sa tentative de preuve d'invariance.

3.1 L'affaiblissement

Supposons que l'on dispose d'une règle d'affaiblissement pouvant s'appliquer sur une partie Δ du contexte : Si le séquent à prouver est de la forme $\vdash \nu Bt, \Delta, \Gamma$ Il faut alors choisir pour chaque formule de Δ si on doit la garder dans l'invariant. Sans induction, on peut décider de n'utiliser l'affaiblissement que lorsqu'on sait qu'on a besoin d'éliminer une formule, par exemple juste avant une règle d'axiome. Ici, la preuve d'invariance n'utilise plus le contexte de façon monotone et le moindre affaiblissement ou la moindre surcharge du contexte peut prendre une importance cruciale

En particulier, la règle d'induction (ν) ne commute pas avec la règle d'affaiblissement. Non seulement certaines preuves ne fonctionnent pas en prenant l'intégralité du contexte car elles demandent de prouver des formules inutiles lors de l'utilisation de l'hypothèse d'induction, mais en plus, avoir un invariant plus court accélère la preuve de son invariance.

Il devient dès lors nécessaire d'effectuer la preuve de l'invariance tout en déterminant à la volée quelles sont les hypothèses à garder et lesquelles sont à rejeter. On peut s'épargner des morceaux de preuves inutiles, par contre cela ne respecte plus totalement le principe de focalisation de la recherche de preuve.

Pour chaque formule ϕ de Δ on crée une méta-formule $?\phi$, qui selon le déroulement de la recherche de preuve, sera amenée à être remplacée par $\bot$ ou bien ϕ elle-même.
On reprend le même invariant

$$S := \lambda x.\exists \Sigma'.x = t^{\Sigma'} \otimes (\Gamma^{\bot})^{\Sigma'} \otimes (?\Delta)^{\bot})^{\Sigma'}$$

On peut reprendre la recherce de preuve directement à partir de $\vdash BSt, ?\Delta, \Gamma$. $?\Delta$ contient les occurences positives de ces méta-formules, et, par monotonicité de B, BSx contient les occurences négatives.

La procédure se déroule alors ainsi : une focalisation ou l'utilisation d'une règle d'axiome sur une occurence positive de $?\phi$ correspond à une vraie utilisation de ϕ et a pour conséquence de ramener $?\phi$ à ϕ de manière globale dans toute la preuve. Par contre les séquents $\vdash ?\phi^{\bot}, \Gamma$, qui sont obtenus pendant l'utilisation d'une hypothèse d'induction doivent être mis en attente tant qu'une occurence positive de $?\phi$ n'a pas été utilisée, ou bien jusqu'à la fin de la preuve.

Si la preuve se termine (c'est à dire si tous les séquents encore présents sont de la forme $\vdash ?\phi^{\bot}, \Gamma$) ,alors on peut ramener tous les $?\phi$ sur $\bot$. Comme $?\phi$ n'a jamais eu le focus, le remplacer par $\bot$ n'enlève rien à la validité de la dérivation, ainsi tous les séquents restants sont de la forme $\vdash, \top, \Gamma$, et donc sont résolus.

Il est tout à fait possible, lorsqu'il y a des inductions imbriquées, qu'une formule accumule plusieurs marqueurs ?. Une utilisation d'une formule après plusieurs inductions imbriquées implique que cette formule est vraiment utilisée dans chaque profondeur d'induction.

Lorsqu'on laisse de coté un séquent avec une occurence négative d'un $?\phi$, on perd la focalisation qu'on avait sur cette formule. Dans le cas où on est amené à dégeler cette formule et à reconsidérer ce séquent, un échec dans la preuve de ce séquent fait échouer la focalisation qui a forcé à reconsidérer le séquent, tandis que la procédure normale aurait fait échouer la focalisation sur l'appel de l'hypothèse d'induction. L'échec ici est du à plusieurs focalisations différentes simultanées, celle qui a amené le séquent lui-même, et celles qui ont dégelé les formules $?\phi^\perp$ du séquent.

Parallèlement, si la preuve d'un séquent $\vdash ?\phi^\perp, \Gamma$ échoue au moment de l'utilisation d'une occurence positive de ϕ, on peut prématurément ramener $?\phi$ sur $\perp$. N'importe quelle utilisation ultérieure de ϕ aurait pour issue le même échec, éventuellement plus rapidement si d'autres méta-formules ou méta-variables ont été spécifiées entre-temps.

Le problème que pose la règle d'affaiblissement est donc difficile à résoudre et nous éloigne du principe de focalisation. Je n'ai pas implémenté cette méthode en **Taci** , l'architecture actuelle des tactiques n'y étant pas adaptée et manquant de souplesse. Une implémentation permettrait de vérifier si il y a bien des gains en efficacité tant en rapidité qu'en nombre de formules prouvables.

3.2 Métavariable ou métaformule

Pour implémenter cette procédure, il est possible d'utiliser le mécanisme de métavariable et d'unification déjà connu et utilisé. Pour prouver un but de la forme $\exists x.P\,x$, le prouveur introduit une métavariable X qu'il va chercher à spécifier au fur et à mesure qu'il cherche une preuve de P.

Ici, on peut utiliser des métavariables pour simuler cette inférence d'invariant, à condition de pouvoir préciser le comportement du prouveur avec ces métavariables. Avant d'utiliser l'induction on remplace le séquent

$$\vdash \nu Bt, \Delta, \Gamma$$

par

$$\vdash \exists X_\Delta(\nu Bt \parr ?\Delta \parr \Gamma)$$

où pour chaque formule ϕ de Δ, on introduit une variable existentielle X_ϕ. $?\phi$ dénote $X_\phi = true \otimes \phi$, et les occurences négatives $?\phi^\perp$ dénotent $X_\phi \neq true \parr \phi^\perp$.

En reprenant le discours ci dessus, une focalisation sur une occurence positive de $?\phi$ amène le prouveur à chercher une preuve de $\vdash X_\phi = true$ et donc à instancier la métavariable X_ϕ en $true$. En revanche, lors de l'apparition d'un séquent $\vdash X_\phi \neq true, \Gamma$ le prouveur doit attendre d'avoir précisé X_ϕ avant de chercher à prouver Γ, au cas où la variable serait instanciée en autre chose que $true$, auquel cas il n'y a pas d'unificateur possible et le séquent est résolu par la règle $(\neq)$.

En outre, lorsque, après instanciation de X_ϕ en *true*, la preuve de $\vdash \Gamma$ est un échec, on aimerait pouvoir s'en rappeler lors des tentatives ultérieures équivalentes.
Ce mécanisme de mise en attente d'un séquent ainsi que l'utilisation de l'échec de la preuve d'un séquent ne sont pas des principes propres à l'inférence d'invariant mais à l'inférence de variables existentielles par unification en général. Si on peut améliorer les principes d'inférence par unification, alors on peut se retrouver sans effort supplémentaire avec une meilleure inférence d'invariant.

3.3 Autres raffinements

La même démarche est envisageable pour d'autres parties de l'invariant. Rappelons l'invariant :

$$S := \lambda x.\exists \Sigma.x = t^\Sigma \otimes (\Gamma^\perp)^\Sigma$$

Ainsi que la preuve de la prémisse de droite après la règle d'induction :

$$\dfrac{\dfrac{\dfrac{}{\Sigma \vdash \Gamma^\perp, \Gamma}\ init \quad \dfrac{}{\Sigma \vdash t^\Sigma = t^\Sigma}}{\Sigma \vdash t^\Sigma = t^\Sigma \otimes (\Gamma^\perp)^\Sigma, \Gamma^\Sigma}\ \otimes}{\Sigma \vdash \exists \Sigma'.t^\Sigma = t^{\Sigma'} \otimes (\Gamma^\perp)^{\Sigma'}, \Gamma^\Sigma}\ \exists$$

Après avoir considéré l'affaiblissement de Γ, on a deux autres endroits où on peut vouloir faire des modifications. On peut vouloir affaiblir la conjonction $x = t^\Sigma$, et on peut dissocier des occurences différentes d'une variable de Σ en introduisant des variables existentielles différentes pour des occurences différentes d'une même variable, ce qui revient à relaxer les liens entre les termes appliqués au point fixe et le reste de la formule.

Par exemple, si on veut faire une induction sur un entier n, cela revient à décider quelles occurences de n vont varier pendant l'induction et lesquelles peuvent être considérées comme constantes, puis pour les autres variables, quelles sont les occurences d'une variable qui doivent être les mêmes, mais aussi quelles occurences d'une variable peuvent se permettre d'être dissociées pendant la récurrence.

Je pense que parmi ces questions, on peut peut-être répondre à certaines d'entre elles en cherchant la preuve de l'invariance à la volée, de la même manière qu'on peut décider des hypothèses à affaiblir. Cette preuve d'invariance, d'une manière générale, demande de faire très attention à ce que l'on fait. Alors qu'auparavant, on pouvait se permettre de ne pas chercher des preuves linéaires, d'affaiblir au dernier moment parcequ'on a pas choisi exactement les hypothèses qu'il fallait, ou alors d'ouvrir un quantificateur un peu trop tôt alors qu'on a un bout de preuve quelquepart qui ne dépend pas de la variable introduite, et ainsi de suite... La preuve d'invariance demande de faire attention à ces choses là, et au lieu de bloquer et d'échouer notre preuve parcequ'on est en train de faire une preuve un peu trop grossière, il faut analyser la preuve que l'on est en train de faire plus en détail pour être prêt à introduire les petites modifications qui vont donner un invariant correct.

Enfin, une dernière possibilité, lorsqu'on travaille en logique intuitionniste, qui peut être rapprochée de l'interprétation abstraite en général, est de renforcer la formule G à prouver par induction en y rajoutant une métaformule complète : passer de G à $G \,\&\, \Psi$.

Bien sûr il faut alors donner des directives précises sur à la fois quelles genres de formules on s'attend à inclure à Ψ, c'est à dire pour quelles formules P faut-il rendre la preuve du séquent Ψ, P, Γ triviale ; et de quelle manière on renforce Ψ, la manière la plus immédiate et la moins approximative étant de remplacer Ψ par $\Psi \otimes P^{\perp}$. On peut ainsi mettre en place une méthode d'inférence d'invariant similaire à celle qu'on connaît déjà en interprétation abstraite.

4 Les types de données

4.1 La notion de progrès

Le choix de l'invariant n'est pas la seule question qui se pose à un prouveur automatique. Il y a trois choix possibles lorsqu'on se trouve devant un νB :

- Le geler
- Faire une induction
- Faire un déroulage

Une question tout aussi importante est de savoir décider pour quels points fixes il est préférable de faire induction, et pour lesquels il est préférable de faire un déroulage. Une réponse est de repérer les points fixes qui correspondent à du *calcul* et moins à de la logique.

En arithmétique, on utilise des points fixes du type :

Nat $x := (x = 0) \oplus (\exists x'.x = sx' \otimes$ Nat $x')$
Plus $x\ y\ z := (x = 0 \otimes y = z) \oplus (\exists x'.x = sx' \otimes \exists z'.z = sz' \otimes$Plus $x'\ y\ z')$
Mult $x\ y\ z := (x = 0 \otimes z = 0) \oplus$
$(\exists x'.x = sx' \otimes \exists z'.\mu$Plus $y\ z'\ z \otimes$ Mult $x'\ y\ z')$

On définit les fonctions par étude de cas en reproduisant le matcher présent dans Nat. Les types de données usuels (les entiers, les listes, les arbres..) ainsi que les fonctions qu'on définit par induction par-dessus sont toujours de cette forme. Les types de données sont construits sur un matcher synchrone, les fonctions sur ce type de données reprennent ce matcher et en sont seulement une extension.

Le principe de progrès est de toujours dérouler un point fixe qui est un matcher sur un argument qui est un terme commençant par un constructeur, par exemple, s (s x), cons y nil, node x (node $y\ z$)... Cela revient à utiliser la définition du point fixe ou des fonctions dès qu'on sait quelque chose sur l'argument principal. Dans la plupart des cas, cela permet de gagner de l'information sur des termes plus simples. Ici, dérouler un point fixe systématiquement est comme utiliser un système de réécriture de formules :

μNat $0 \rightarrow \top$
μNat (s x) $\rightarrow \mu$Nat x
μPlus $0\ y\ z \rightarrow y = z$
μPlus (s x) $y\ z \rightarrow \exists z'.\ z =$ s $z' \otimes \mu$Plus $x\ y\ z'$

De même par exemple, sur des listes

$\mu\text{List nil} \to \top$
$\mu\text{List (cons } x\ y) \to \mu\text{List } y$
$\mu\text{Length nil } x \to x = 0$
$\mu\text{Length (cons } x\ y)\ z \to \exists z'.\ z = \text{s } z' \otimes \mu\text{Length } y\ z'$

On peut même décider d'aller plus loin, par exemple dans le cas de Length, le point fixe peut être considéré à la fois comme un matcher de liste sur le premier argument et un matcher de nat sur le deuxième :

$\text{Length } l\ x := (l = \text{nil} \otimes x = 0) \oplus$
$(\exists y. \exists l'. \exists x'. l = \text{cons } y\ l' \otimes x = \text{s } x' \otimes \text{Length } l'\ x')$

En réarrangeant les sous-formules à l'intérieur de Length (commutativité de $\otimes$, de $\exists$, et réarrangement des quantificateurs), on peut écrire Length comme étant une fonction sur ses chacun de ses arguments. Ainsi on pourra utiliser des règles de réécriture supplémentaires en déroulant systématiquement sur le deuxième argument :

$\mu\text{Length } l\ 0 \to l = \text{nil}$
$\mu\text{Length } l\ (\text{s } x) \to \exists y. \exists l'. l = \text{cons } y\ l' \otimes \mu\text{Length } l'\ x$
$\mu\text{Length nil } 0 \to \top$
$\mu\text{Length (cons } x\ y)\ (\text{s } z) \to \mu\text{Length } y\ z$

Généralement, les systèmes qui ont les fonctions comme objets de première classe (à la différence de prédicats ou d'ensembles), ont ces règles de réécriture (par exemple, la tactique *simpl* fait ces simplifications dans le système Coq). Dans le cas de Length, avoir un moyen de reconnaître le fait que le prédicat est du pattern-matching sur ses deux arguments en même temps est quelque chose d'un peu plus fort qu'on aurait pas simplement dans ces systèmes.

Ces réécritures systématiques sont très utiles pour éviter de poser des choix au prouveur, pour avoir la certitude qu'on ne prend aucun risque en effectuant ces déroulages de manière asynchrone, et que ces déroulages sont utiles puisqu'ils nous donnent des informations sur les variables présentes dans le séquent. Cependant, il est possible de toujours gagner de l'information sur un terme sans jamais avoir à s'arrêter : avec un point fixe simple et des données particulières, il est possible que la phase asynchrone ne termine jamais :
On considère encore le point fixe μLength.
Supposons qu'on dispose des deux hypothèses $\mu\text{Length } l\ n$ et $\mu\text{Length } l\ (\text{s } n)$ simultanément.

Lors d'une phase asynchrone, on va appliquer la règle sur Length l (s n) dans la deuxième hypothèse, ce qui va nous apprendre que l s'écrit en fait (cons $x\ l'$).
Toujours en phase asynchrone, l est réécrit dans la première hypothèse.
On peut alors appliquer la règle sur Length (cons $x\ l'$) n dans la première hypothèse, ce qui va nous apprendre que n s'écrit en fait (s n').
n est alors réécrit en (s n') dans la deuxième hypothèse, et on se retrouve au point de départ.

En pratique, cette situation est assez rare, et les cas où l'utilisation systématique du progrès diverge ne sont pas aussi simples que sur cet exemple. Cette situation n'est pas due au fait d'avoir un prédicat qui utilise la notion de progrès sur ses deux arguments à la fois : on pourrait très bien définir deux prédicats Length1 et Length2 qui progressent chacun sur un seul argument et reprendre l'exemple précédent.
D'autre part, sur cet exemple, on a une contradiction, et on pourrait prouver $\perp$ par induction assez simplement, mais il existe des situations, avec d'autres points fixes, où une phase asynchrone peut avoir un comportement infini sans pour autant avoir une contradiction dans les hypothèses.

On a donc choisi dans l'implémentation de ce principe dans **Taci** , de limiter le nombre de déroulages effectués selon ce critère au sein d'une phase asynchrone. Sans la notion de progrès, l'utilisateur autant que le prouveur sont gênés par les calculs à faire à la main. C'est un mécanisme indispensable pour trouver facilement des preuves même simples.

4.2 La structure des matchers

Une étude sur les types de données peut justifier pourquoi utiliser un déroulage systématique asynchrone sur certains points fixes est possible.

La partie commune aux points fixe Nat, Plus et Mult, qui explique pourquoi ils fonctionnent si bien est le **matcher** qui définit le point fixe Nat. Un **matcher** est un connecteur synchrone M qui prend une variable x en argument et n formules $\phi_1 \cdots \phi_n$. On note $M_x(\phi_1, \phi_2 \cdots \phi_n)$ l'application de M. Un **matcher** est particulièrement intéressant lorsqu'il avec les connecteurs synchrones et lorsqu'il s'absorbe lui-même : Par exemple, pour un **matcher** d'arité 2, on veut avoir $\forall x. \forall \phi_1, \phi_2, \phi_3, \phi_4$:

$$M_x(\phi_1, \phi_2) \otimes M_x(\phi_3, \phi_4) \leftrightarrow M_x(\phi_1 \otimes \phi_3, \phi_2 \otimes \phi_4)$$

$$M_x(\phi_1, \phi_2) \oplus M_x(\phi_3, \phi_4) \leftrightarrow M_x(\phi_1 \oplus \phi_3, \phi_2 \otimes \phi_4)$$

$$\exists y. M_x(\phi_1, \phi_2) \leftrightarrow M_x(\exists y. \phi_1, \exists y. \phi_2)$$

$$M_x(\phi_1, M_x(\phi_2, \phi_3)) \leftrightarrow M_x(\phi_1, \phi_3) \leftrightarrow M_x(M_x(\phi_1, \phi_2), \phi_3)$$

La commutation avec $\otimes$ et avec M_x n'est pas triviale et est valable lorsque x ne peut pas être décomposé simultanément dans deux branches différentes de M_x.Le reste est du à la synchronicité d'un **matcher** .
En outre, on a l'équivalence, pour toute formule ψ,

$$M_x(\phi_1, \phi_2) \otimes \psi \leftrightarrow M_x(\phi_1 \otimes \psi, \phi_2 \otimes \psi)$$

Dualement, $M^\perp$ commute alors avec tous les connecteurs asynchrones.

Par exemple, le **matcher** utilisé par Nat est
$M := \lambda x. \lambda \phi_1. \lambda \phi_2. (x = 0 \otimes \phi_1) \oplus (\exists x'. x = \mathrm{s}\ x' \otimes \phi_2)$
La variable x' introduite dans la seconde branche joue un rôle particulier dans ϕ_2.

Un **type de donnée** est un point fixe dont le corps est un **matcher** complété seulement par des occurences récursives du point fixe.

Le corps du point fixe Nat est
$Nat := \lambda nat.\lambda x.M_x(\top, natx')$
et c'est donc un **type de donnée** .

Les règles d'induction et de déroulage des points fixes font toujours apparaître le matcher M_x en tête de la formule considérée. Ainsi, une fois qu'on veut faire un déroulage ou une induction sur un $\nu Nat^\perp x$ (ou autre point fixe construit sur le matcher de Nat), on fait apparaître un $M_x^\perp$ en tête de la formule. En faisant jouer les équivalences ci-dessus on peut alors anticiper les utilisations des autres points fixes présents dans le contexte derrière des connecteurs asynchrones et décider de factoriser le travail de $M_x^\perp$ en les faisant remonter en tête des formules, puis en regroupant toutes ses occurences en une seule (avec la commutativité du $\parr$). On ne perd donc rien à systématiquement dérouler tous les points fixes $\mu Natx$ accessibles pendant la phase asynchrone.

Le critère de progression utilisé dans la section précédente est un moyen statique de forcer cette optimisation. En pratique, évaluer un matcher M_x va donner des informations sur x (on dit qu'il y a progrès) au sens où il identifie le constructeur en tête de x. Dès lors, on sait que l'évaluation des matchers suivants sera facile et n'entrera que dans une seule branche du matcher : il n'y a pas de risque de devoir faire une étude de cas supplémentaire.

Une autre conséquence intéressante de la synchronicité du matcher, est que si les points fixes basés sur M restent totalement synchrones (en considérant qu'un point fixe dont le corps est synchrone est synchrone), alors on a pour ces points fixes l'équivalence $\mu Bt \leftrightarrow \mu Bt \otimes \mu Bt$. On peut alors l'utiliser pour effectuer des inductions sans pour autant faire disparaître le point fixe sur lequel on induit en le dupliquant préalablement. Dès lors, la règle d'induction devient de la forme

$$\frac{BSt, \Gamma}{\Gamma}$$

Dans le cas où $B\ S\ t$ est un $M_t^\perp(\phi_1 \cdots \phi_n)$, on peut regrouper le $M_t^\perp$ avec ceux qui sont prêts à être déroulés dans Γ (en particulier, le point fixe duquel on est parti en fait partie). De ce qui est dans les ϕ_i, tout ce qui n'était pas une occurence récursive de νB devient redondant, la seule formule finalement importante est l'hypothèse d'induction :
La seule différence entre faire une induction sur Plus x et faire une induction sur Nat x se trouve dans l'hypothèse d'induction, plus particulièrement les égalités liant les arguments de l'appel récursif à l'environnement sur les arguments du point fixe. Choisir de faire une induction sur Nat plutôt que sur Plus donne alors une hypothèse d'induction plus facile à utiliser car Nat a moins d'arguments que Plus, sans pour autant rendre la preuve plus difficile à un autre endroit. Ainsi, en arithmétique, faire une induction sur Plus x ou Mult x est la même chose qu'une induction sur Nat x, mais en un peu plus difficile.
De même, par le biais du progrès, faire seulement un déroulage sur Nat x va avoir pour conséquence de dérouler les Plus x et Mult x présents dans le contexte.

Une optimisation importante à en tirer est qu'il est donc possible d'interdire totalement

les règles d'induction et de déroulage non asynchrone sur les points fixes Plus x et Mult x, pourvu qu'on dispose de Nat x dans le contexte.
Au besoin, il est immédiat de montrer que Nat est impliqué par les autres points fixes, donc il n'est pas difficile de le rajouter.
Les points fixes Plus et Mult n'interviennent alors plus que sous forme de calculs, toute la logique de point fixe étant ramenée sur le type de donnée Nat seulement.

On a donc montré que sous certaines conditions à vérifier sur les matchers, on peut restreindre l'utilisation de la règle d'induction aux points fixes qui correspondent à un **type de donnée** construit sur un **matcher** . Les fonctions définies sur ces types de données sont certes des points fixes, mais elles ne complexifient pas la recherche d'une preuve. Ces fonctions, par le biais de la notion de progrès, sont automatiquement réduites et calculées lorsqu'elles le peuvent. Elles ne représentent donc plus qu'une notion de calcul et n'ajoutent plus aucun choix dans le raisonnement ou la recherche de preuves.

Le lien entre Nat et les fonctions définies par récurrence peut être caractérisé par le fait que le séquent $\vdash \nu Plus^{\perp} xyz, \mu Nat x$ est prouvé immédiatement en une induction, une phase asynchrone, puis une focalisation sur μNat.

Dans **Taci** , c'est encore à l'utilisateur de préciser quels arguments de quels points fixes il faut examiner pour décider du progrès ou non d'un point fixe. On peut cependant envisager que ce rôle pourrait être en partie laissé au prouveur, les conditions requises sur la structure des points fixes pouvant être déterminées au moins statiquement dans certains cas, et peut-être après des preuves rapides de lemmes dans des cas plus complexes, comme la fonction longueur des suites qui peut être vue comme une fonction de ses deux arguments.

4.3 La Skolémisation

On a montré que l'utilisation du progrès permettait de simuler une recherche de preuve où on déplaçait la position des matchers dans une formule pour les rendre au plus haut niveau possible.
Cela peut être rapproché d'une recherche de preuve qui travaille modulo certaines équivalences de formules, dans le cas de progrès, il s'agissait de la navigation des matchers dans une formule.

Lorsqu'on travaille avec des fonctions, un problème se pose rapidement qui nous pousse à essayer de Skolemiser les fonctions : Supposons que je veuille montrer $(\Gamma, \mu\text{Nat } x) \to \exists z.\mu\text{Plus } x\ y\ z \otimes \phi\ z$. Vouloir donner un témoin z ne va jamais marcher si il ne peut pas être obtenu à partir de la signature du séquent et si Γ ne donne rien sur la somme de x et y. L'autre solution est une induction sur x, mais peut-être que la preuve de ϕ ne se fait pas par induction sur x. S'il en reste là, le prouveur reste coincé sur ce connecteur $\exists z.\mu\text{Plus } x\ y\ z \otimes \phi$.

Si on a prouvé que l'addition était une fonction on connait alors l'existence et l'unicité de z. Ce connecteur prend alors la forme $C(\phi) := \exists x.P\ x \otimes \phi$, où P est un prédicat qui a la propriété d'être un singleton :

$$\forall x \forall y. Px \to Py \to x = y$$

$$\exists x. Px$$

Si on a la possibilité de dupliquer l'hypothèse $P\ x$, alors ces deux formules peuvent montrer facilement que C permute avec les connecteurs $\parr$, $\&$, $\otimes$, $\oplus$, ainsi qu'avec les constantes 1, 0, $\top$, $\bot$, et les quantificateurs $\forall$ et $\exists$ qui ne lient pas de variable présente dans P. En conséquence, on peut prouver les formules

$$\vdash C(\phi), C(\phi^\bot)$$

$$\vdash C^\bot(\phi), C^\bot(\phi^\bot)$$

Ce qui indique que l'opérateur C est équivalent à son dual $C^\bot(\phi) = \forall z.Pz \to \phi$.

Ce connecteur est donc plus structurel que logique. On pourrait par exemple le repousser au fond des formules jusque dans les termes moyennant quelques difficultés au niveau de l'application des points fixes, et obtenir un comportement proche d'une logique qui remplacerait directement la variable z par un terme $\epsilon(P)$.
On peut faire une comparaison intéressante avec le quantificateur générique ∇ : Lorsqu'on ajoute ce quantificateur à la logique μMALL, le quantificateur ∇ est lui aussi auto-dual, et peut également être défini structurellement [3] jusqu'à être presque éliminé des formules.

Le comportement que je propose est de laisser le soin au prouveur de choisir à loisir la polarité du connecteur, de manière à ne jamais devoir fournir un témoin, mais plutôt d'avoir le droit d'introduire l'élément du singleton. Sans principe d'induction, cela aurait un intérêt mitigé car il suffirait de choisir correctement les polarités de chaque occurence du connecteur au moment où l'utilisateur écrit le théorème à prouver et il n'y aurait rien à faire.
Au moment où on utilise le principe d'induction, quelle que soit la façon dont ont été attribuées les polarités de ces connecteurs, il duplique le contexte Γ en un Γ et un $\Gamma^\bot$. Pour pouvoir travailler facilement, il faut donc repérer la présence de ces connecteurs dans $\Gamma^\bot$ et systématiquement passer à leur version duale.

4.3.1 Exemple

Dans cette exemple on écrira plus et mult au lieu de μMult ou μPlus, et on travaillera dans des séquents à deux cotés, pour plus de clarté.
Supposons qu'on possède déjà des résultats de fonctionnalité sur l'addition et qu'on veuille montrer le théorème

$\vdash \forall xyzt.\ \text{mult}\ x\ y\ z \to \text{plus}\ x\ z\ t \to \text{mult}\ x\ (\text{s}\ y)\ t$

Ce théorème est impossible à montrer par un prouveur qui n'utilise pas d'une manière ou d'une autre la fonctionalité de l'addition. Utiliser l'auto-dualité du connecteur décrit plus haut élimine totalement la difficulté et rend la preuve faisable.
Développer cette formuler en une phase asynchrone donne :

$\text{mult}\ x\ y\ z, \text{plus}\ x\ z\ t \vdash \text{mult}\ x\ (\text{s}\ y)\ t$

La seule chose à faire est une induction sur nat x. Le cas où x est nul ne posant pas de difficulté, on s'intéresse directement au cas inductif :

$H, \text{mult}\ (\text{s}\ x)\ y\ z, \text{plus}\ (\text{s}\ x)\ z\ t \vdash \text{mult}\ (\text{s}\ x)\ (\text{s}\ y)\ t$

H est l'hypothèse d'induction $\forall yzt.\ \text{mult}\ x\ y\ z \to \text{plus}\ x\ z\ t \to \text{mult}\ x\ (\text{s}\ y)\ t$

C'est dans cette hypothèse d'induction qu'on doit remarquer qu'il y avait un connecteur autodual :
H est équivalent à $\forall yz.$ mult x y $z \rightarrow \forall t.$ plus x z $t \rightarrow$ mult x (s y) t
On sait que $P(t) =$ plus x z t est un singleton car on a nat x impliqué par nos hypothèses. On réécrit donc H en

$H = \forall yz.$ mult x y $z \rightarrow \exists t.$ plus x z $t \otimes$ mult x (s y) t.
A partir de là, la preuve continue rapidement : les points fixes présents dans le contexte progressent, et en une phase asynchrone, on obtient :

H, plus y z' z, mult x y z', plus x z $t' \vdash \exists u.$ plus (s y) u (s t') $\otimes$ mult x (s y) u
On focalise sur l'hypothèse d'induction H en utilisant mult x y z' :

plus x z' u, mult x (s y) u, plus y z' z, plus x z $t' \vdash \exists u.$ plus (s y) u (s t') $\otimes$ mult x (s y) u
On focalise sur le but en utilisant mult x (s y) u :

plus x z' u, plus y z' z, plus x z $t' \vdash$ plus (s y) u (s t')
Ce qui est prouvable en utilisant la commutativité et l'associativité de l'addition.

Exemple de théorèmes résolus

Dès qu'on dispose du critère de progrès pour effectuer des déroulages de manière asynchrone, on peut prouver facilement tous les théorèmes simples sur l'addition. Cette procédure est implémentée dans la tactique de preuve automatique avec **Taci** , elle est alors suffisante pour montrer de nombreux théorèmes simples :
A propos de l'addition :

- $\forall xyz$ plus x y $z \rightarrow$ nat x
- $\forall xy$ nat $x \rightarrow \exists z$ plus x y z
- $\forall xyzt$ plus x y z, plus x y $t \rightarrow z{=}t$
- $\forall xyz$ nat y plus x y $z \rightarrow$ plus y x z
- $\forall xyztuv$ plus x y t,plus y z u,plus t z $v \rightarrow$ plus x u v

- $\forall xyz$ plus x y $z \rightarrow x \leqslant z$
- $\forall x_1x_2yz_1z_2$ plus x_1 y z_1,plus x_2 y z_2, $x_1 \leqslant x_2 \rightarrow z_1 \leqslant z_2$

On définit la division par 2 :
Half x $h :=$ (x=0 $\otimes$ h=0) $\oplus$ (x=s 0 $\otimes$ h=0) $\oplus$ ($\exists$ y h'. x=s s $y \otimes$ Half y h')

- $\forall x$ nat x $\rightarrow \exists y$ half x y

Ce résultat se montre en deux inductions imbriquées (une sur nat et l'autre sur half), mais il devrait pouvoir être résoluble par induction forte. Des théorèmes sur des listes : On définit les prédicats prefix, suffix puis sublist comme combinaison des deux

- $\forall l$ list $l \rightarrow$ prefixe l l
- $\forall l$ list $l \rightarrow$ suffixe l l
- $\forall l$ list $l \rightarrow$ sublist l l

Le théorème sur sublist n'est pas prouvable sans utiliser les lemmes, c'est un cas où le contexte pris au moment de la génération de l'invariant n'est pas adéquat. Toujours à propos de listes, en définissant les listes d'entiers puis le tri par insertion, on peut montrer que le résultat du

tri est trié et que la liste triée possède les mêmes élément que la liste de départ.

Dans un tout autre domaine, on peut spécifier un jeu de morpion à base de listes à 9 éléments.

```
%  winner x l est vrai si il y a un alignement de 3 x dans la grille l

winner x l := exists a, b, d, e, f, g.
l = c x (c x (c x (c a (c b (c d (c e (c f (c g nil)))))))));
l = c a (c b (c d (c x (c x (c x (c e (c f (c g nil)))))))));
...
l = c b (c d (c x (c a (c x (c g (c x (c e (c f nil))))))))

%  move x l k décrit par induction sur la liste les coups possible par x de l vers k

move x l k := exists l0. l = c n l0, k = c x l0) ;
 (exists l0\ k0\ a\ l = c a l0, k = c a k0, move x l0 k0)

%  wins p o l indique si le joueur p a une stratégie gagnante contre le joueur o
%  lorsque o joue sur la grille l

wins p o l := winner p l ; pi l0\ move o l l0 =>
 exists l1\ move p l0 l1, wins p o l1

%  flip définit une symétrie axiale de la grille

flip l1 l2 := exists a\ b\ d\ e\ f\ g\ h\ i\ j\
 l1 = c a (c b (c d (c e (c f (c g (c h (c i (c j nil)))))))),
 l2 = c d (c b (c a (c g (c f (c e (c j (c i (c h nil)))))))))
```

Taci peut alors montrer que le jeu est effectivement symétrique par l'opération *flip*. En particulier il montre que si il existe une stratégie gagnante sur une grille donnée alors il en existe encore une sur la grille symétrique.
Pour prouver ce genre de résultat, le model-checking a besoin d'explorer toutes les parties possibles. Avec l'induction, on peut maintenant montrer ce genre de résultat rapidement.

5 Conclusion et travaux futurs

L'utilisation de l'induction dans une recherche de preuve focalisée a soulevé de bonnes questions. Les techniques utilisées comme l'invariance d'invariant en utilisant des métavariables demandent d'avoir une description et une compréhension approfondie du principe d'unification et d'inférence des variables existentielles par un prouveur automatique. Ce principe n'est généralement pas décrit dans la logique mais c'est le prouveur qui la développe à l'extérieur de la logique.

L'inférence des variables existentielles révèle des problèmes comme la non commutativité des connecteurs. Selon la branche que choisit un prouveur, la preuve peut partir dans deux

directions radicalement différentes, tant bien que l'utilisateur est parfois forcé de donner son théorème en permutant les formules de quelques connecteurs.

Lorsqu'on veut se servir de ce principe afin d'inférrer un invariant plus finement que grossièrement supposer que le contexte va convenir, pouvoir décrire l'ordre dans lequel le prouveur doit explorer les branches d'une preuve devient une nécessité. En outre il suggère de mettre en place une utilisation de l'échec d'une preuve. Il est donc bon de se pencher plus profondément sur le mécanisme d'inférence de métavariable. Bien que difficile à mettre en place, l'inférence est un mécanisme qui permet de trouver plus de preuves sans tellement rajouter de recherche inutile, déterminer quelles peuvent être ses utilisations donne toujours un avantage théorique à une stratégie de recherche de preuve.

La logique $\mu MALL^{=}$ est déjà assez expressive et son principe d'induction devrait lui donner les moyens de simuler au niveau des preuves assez fidèlement les méthodes de systèmes différents. Le prochain niveau de simulation est alors de chercher à simuler la recherche de preuve elle-même. Peut-on par exemple avoir une tactique qui cherche à typer un programme de la même manière que le ferait un système de type ? Il y a aussi de l'inspiration à prendre du coté de l'interprétation abstraite relativement à la génération d'invariants dans des domaines spécifiques.

Parallèlement, la théorie des matchers et des types de données permet de retrouver une partie du comportement des systèmes basés sur de la réécriture ou des systèmes disposant d'une vraie notion de fonction.

Les techniques que j'ai présentées reposent sur des notions d'équivalence de formules. D'une certaine manière, la focalisation est une stratégie qui rend indiscernable au prouveur certaines familles de formules. Par exemple la commutation de deux quantifications universelles, ou bien le réarrangement des formules dans une grosse conjonction, mais aussi les mélanges de ces deux connecteurs, permettent d'écrire des formules différentes mais équivalentes. La focalisation fait en sorte que le prouveur ne voit pas ces différences.

En continuant dans cette voix, on pourrait vouloir un prouveur qui ne fasse plus de différence entre $a + b$ et $b + a$ quand il a prouvé la commutativité de l'addition, en utilisant une technique similaire à celle que j'ai décrite pour gérer les formules skolémisées. Finalement, ces lemmes que l'on voudrait que le prouveur utilise, sont utilisés en modifiant directement la tactique de recherche de preuve. En quelque sorte le prouveur est en train de modifier lui même sa stratégie de recherche au fur et à mesure qu'il prouve des résultats de base sur les prédicats et les fonctions manipulées.

Une approche naïve d'utilisation de lemmes, comme par exemple rajouter les lemmes qu'on pense connexe au théorème à prouver systématiquement en hypothèses, est assez inefficace en raison du nombre de nouveaux choix que le prouveur doit considérer. Savoir utiliser les résultats déjà prouvés de manière plus subtiles et plus profondes, c'est à dire en sachant à l'avance comment ils doivent être utilisés est une solution plus attirante même si elle nécessite du travail pour rendre le prouveur capable de l'utiliser de cette manière.

La quantification générique ∇ introduite par Tiu [10] permet de formaliser facilement les spécifications logiques qui définissent des langages de programmation, ou des calculs comme le π-calcul ou le λ-calcul, ce qui ouvre la voie à la preuve automatique de théorèmes issus de ces théories. Plus les langages et calculs considérés sont complexes, plus les preuves comme la

sureté du typage sont longues et difficiles à suivre bien que leur principes restent les mêmes. En réponse au POPLMark Challenge, on voit alors un intérêt à disposer d'une tactique de preuve robuste et d'un prouveur interactif construit sur cette logique.

Références

[1] Anderson and Pfenning. Verifying uniqueness in a logical framework.

[2] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. 1992.

[3] David Baelde. On the expressivity of minimal generic quantification. 2008.

[4] Kaufmann Boyer, Moore. The boyer-moore theorem prover and its interactive enhancement. 1993.

[5] Alwen Tiu Dale Miller. A proof theory for generic judgments. 2005.

[6] Dale Miller and David Baelde. Least and greatest fixed points in linear logic. 2007.

[7] Pfenning and Schürmann. System description : Twelf – a meta-logical framework for deductive systems.

[8] Schürmann and Pfenning. A coverage checking algorithm for lf.

[9] Tac, a generic and adaptable interactive theorem prover, 2008. http ://slimmer.gforge.inria.fr/tac/.

[10] Alwen Tiu. *A logical Framework for reasoning about logical specifications.* PhD thesis.