Elimination des quantificateurs sur les réels pour Coq

Assia Mahboubi[1], Loïc Pottier[2]

1: École normale supérieure de Lyon
46, Allée d'Italie 69364 LYON Cedex FRANCE
amahboub@ens-lyon.fr
2: INRIA Sophia Antipolis, Projet Lemme
2004 route des Lucioles, B.P. 93
06902 SOPHIA ANTIPOLIS Cedex FRANCE
Loic.Pottier@sophia.inria.fr

1. Introduction

Le système Coq[3] permet maintenant de faire des preuves formelles en analyse réelle. Mais les preuves dans ce domaine sont encore très pénibles, car très peu automatisées, en particulier celles qui manipulent les inégalités. Dans le cas des équations et inéquations linéaires, la tactique Fourier permet au système de faire lui-même les preuves, mais cette tactique ne s'applique pas dans le cas de systèmes d'équations et d'inéquations polynômiales. Pourtant il existe pour ce cas des procédures de décision efficaces et simples, au moins en degré faible et avec un nombre de variables ne dépassant pas la dizaine.

Le but du travail qu'on présente ici [1] est d'implémenter en Ocaml[7]une telle procédure sous la forme d'une tactique de Coq. Nous pensons qu'elle pourra être très utile pour les utilisateurs de Coq qui travailleront avec des nombres réels. On peut pour s'en convaincre constater que de très nombreuses preuves faites dans la bibliothèque Reals de Coq reviennent à montrer qu'un système d'inéquations polynômiales n'a pas de solution (par exemple dans les calculs de limites, de continuité).

La méthode qu'on a choisie est une méthode d'élimination des quantificateurs dans les corps réels clos, basée sur le principe de Tarski-Seidenberg, et est exposée dans [1]. Basée sur une démonstration de [6], elle a l'avantage d'être assez simple à programmer, relativement efficace, et de ne faire intervenir que des théorèmes d'analyse réelle élémentaire (essentiellement le théorème des valeurs intermédiaires, prouvé en Coq par [5]).

En pratique la tactique suit les modèles des tactiques Omega et Fourier. On commence par traduire les objets mathématiques de Coq qui interviennent dans les hypothèses et dans le but qu'on veut prouver dans des types de données de Ocaml, puis on effectue le calcul de la procédure de décision en Ocaml. Si celle-ci réussit, on construit alors à partir d'une trace des calculs effectués une preuve dans le formalisme de Coq, qui est ensuite vérifiée par le système.

Si cette tactique s'avère utile, on pourra ensuite envisager d'utiliser la technique de la réflexion pour programmer et vérifier dans Coq lui-même la procédure de décision utilisée (ce qui reviendrait à prouver dans Coq la procédure elle-même alors qu'ici on génère une preuve pour chaque entrée). Mais ce n'est pas encore à l'ordre du jour, d'autant que la programmation de la méthode de [1] a fait apparaître de nombreuses améliorations algorithmiques possibles : l'algorithme n'est de fait pas encore stabilisé.

[1]Ce travail a en grande partie été effectué lors du stage de deuxième année de l'ENS Lyon d'Assia Mahboubi, dans le projet Lemme de l'INRIA Sophia Antipolis, encadré par Loïc Pottier et Marie-Françoise Roy (IRMAR Rennes I). Ce stage est décrit dans [8].

Le plan de cet article est le suivant : on commence par exposer la procédure d'élimination des quantificateurs de [1], qui permet essentiellement de calculer tous les signes possibles d'une famille de polynômes en plusieurs variables, et on donne un exemple d'application de cette méthode. Puis on décrit quelles traces des calculs on utilise. Ensuite, on montre comment utiliser cette trace des calculs pour effectuer l'élimination des quantificateurs universels. Enfin on montre comment reconstruire une preuve Coq à partir de ces traces des calculs.

2. Tableaux des signes d'une famille de polynômes.

Soit $f = \{f_1, \ldots, f_s\}$ une famille de polynômes de $\mathbb{R}[X_1, \ldots, X_n]$. On définit le signe d'un réel par $-$, 0 ou + selon qu'il est négatif, nul ou positif. Un tableau de signes w de f, pour des valeurs $x_1, \ldots, x_{n-1}$ des $n-1$ premières variables, est donné par une subdivision $y_1 < \ldots < y_p$ de $]-\infty; +\infty[$ telle que pour tout intervalle $I \in \{\,]-\infty; y_1[, [y_1],\ ]y_1; y_2[,\ [y_2], \ldots,\ ]y_p; +\infty[\, \}$, pour tout $j \in \{1 \ldots s\}$, et pour tout $x \in I$, $f_j(x_1, \ldots, x_{n-1}, x)$ a pour signe $w(I, j)$. Lorsque le tableau n'a qu'une seule colonne, tous les polynômes de la famille ont un signe constant sur $]-\infty; +\infty[$

Par exemple, un tableau de signes pour $\{X_3^2 + X_1X_3 + X_2,\ X_1^2 - 4X_2\}$ est

	$]-\infty; y_1[$	$[y_1]$	$]y_1; y_2[$	$[y_2]$	$]y_2; +\infty[$
$X_3^2 + X_1X_3 + X_2$	+	0	$-$	0	+
$X_1^2 - 4X_2$	+	+	+	+	+

il donne les signes de ces deux polynômes pour X_3 variant entre $-\infty$ et $+\infty$,
avec $y_1 = \dfrac{-x_1 - \sqrt{x_1^2 - 4x_2}}{2}$, et $y_2 = \dfrac{-x_1 + \sqrt{x_1^2 - 4x_2}}{2}$.

Il n'y a que deux autres tableaux possibles pour ces polynômes :

	$]-\infty; +\infty[$
$X_3^2 + X_1X_3 + X_2$	+
$X_1^2 - 4X_2$	$-$

	$]-\infty; y_1[$	$[y_1]$	$]y_1; +\infty[$
$X_3^2 + X_1X_3 + X_2$	+	0	+
$X_1^2 - 4X_2$	0	0	0

avec $y1 = \dfrac{-x_1}{2}$.

On a donc calculé dans cet exemple le signe du trinôme du second degré en fonction du signe de son discriminant et dans ce cas simple, on sait exprimmer les y_i en fonction des x_j.

Le but de ce qui suit est de calculer tous les tableaux de signes d'une famille f, correspondant aux valeurs possibles des $n-1$ premières variables.

2.1. Cas d'une variable: $\mathbb{Z}[X]$.

On traite ici le cas de polynômes de $\mathbb{Z}[X]$, car ce qui suit s'étend immédiatement à la fois à $\mathbb{R}[X]$ et à $\mathbb{R}[X_1, \ldots, X_{n-1}][X_n]$. En effet, l'outil principal qu'on utilise est la pseudo-division euclidienne.

Supposons les polynômes de f non identiquement nuls et f_s non constant, de degré maximal (si ce n'est pas le cas, tous les polynômes sont constants, leur tableau de signes est trivial).

Soient $a_1, \ldots, a_{s-1}$ les coefficients dominants de $f_1, \ldots, f_{s-1}$ et a_s celui de f_s'. Par pseudo-division euclidienne, on obtient:

$$\forall i, 1 \leq i \leq s-1,\ \ c_i f_s = q_i f_i + g_i,\ deg(g_i) < deg(f_i),\ \ c_s f_s = q_s f_s' + g_s,\ deg(g_s) < deg(f_s')$$

avec $\forall i, c_i > 0$ et c_i divise une puissance de a_i.

Soient w' le tableau de signe de $\{f_1, \ldots, f_{s-1}, f_s', g_1, \ldots, g_s\}$ et $x_1 < \ldots < x_N$ sa subdivision.

Soient $z_1 < \ldots < z_p$ les x_i qui sont racines de $f_1, \ldots, f_{s-1}$ ou f'_s. Des équations de pseudo-division, on déduit qu'en un z_i racine de f_j (resp f'_s), f_s a le signe de g_j (resp g_s). Entre les z_i, le théorème des valeurs intermédiaires donne éventuellement des racines de f_s, une au plus par intervalle, f'_s y étant non nulle et de signe constant.

Par exemple, supposons que z_1 est racine de f_j (soit $w'([z_1],\ j) = 0$), z_2 est racine de f_k (soit $w'([z_2],\ k) = 0$), et que f'_s est positive sur $]-\infty; x_1[$ (soit $w'(]-\infty; x_1[, s) = +$). Alors

- Si $w'([z_1], s+j) = +$ et $w'([z_2], s+k) = -$, alors il existe un unique $y \in]z_1; z_2[$ tel que $f_s(y) = 0$. En effet f'_s ne s'annule pas sur $]z_1; z_2[$, donc comme elle est continue, elle garde un signe constant. Or $g_j(z_1) > 0$ et $g_k(z_2) < 0$, donc $f_s(z_1) > 0$ et $f_s(z_2) < 0$ et comme f_s est une fonction continue strictement monotone sur $]z_1; z_2[$, le théorème des valeurs intermédiaires assure le résultat.
- Si $w'([z_1], s+j) = +$ et $w'([z_2], s+k) \neq -$, alors on peut déduire que f_s ne s'annule pas sur $]z_1; z_2[$, car elle est strictement monotone et continue.
- Si $w'([z_1], s+j) = +$, alors il existe un unique $y \in]-\infty; z_1[$ tel que $f_s(y) = 0$. En effet f'_s est positive en $-\infty$, non nulle avant z_1, $g_j(z_1)$ est positif donc $f_s(z_1)$ aussi. Finalement f_s est un polynôme strictement croissant sur $]-\infty; z_1[$, positif strictement en z_1 et le théorème des valeurs intermédiaires assure le résultat.
- Si $w'([z_1], s+j) \neq +$, alors on peut déduire que f_s ne s'annule pas sur $]-\infty; z_1[$.

On peut voir que tous les autres cas se traitent de la même manière.

Soient $y_1 < \ldots < y_M$ la réunion des racines de f_s données par le théorème des valeurs intermédiaires et des z_i non racines de f'_s. Ces points donnent la subdivision du tableau w. Les signes de $f_1, \ldots, f_{s-1}$ sont tirés directement de w'. Ceux de f_s sont déduits de ceux de f'_s et des g_j dans w'.

En itérant ce processus, on se ramène fatalement à une famille de polynômes constants, dont le tableau de signes est trivial. En effet, en appliquant la transformation qui fait passer de $\{f_1, \ldots, f_s\}$ à $\{f_1, \ldots, f_{s-1}, f'_s, g_1, \ldots, g_s\}$, soit le degré maximum des polynômes décroît, soit il stagne mais alors le nombre de polynômes qui ont ce degré décroît.

2.2. Cas de plusieurs variables: $\mathbb{R}[X_1]...[X_n]$.

La méthode générale est basée sur celle du cas à une variable, en ajoutant le traitement de la nullité éventuelle des coefficients dominants non constants. En effet, les coefficients des polynômes sont maintenant des polynômes en $X_1, \ldots, X_{n-1}$, donc pouvant être nuls ou non selon les valeurs de ces variables.

On effectue les calculs en gardant en mémoire une famille $\mathcal{N}$ de polynômes qu'on suppose non nuls. Ce seront en fait des facteurs des coefficients dominants qui interviennent dans les pseudo-divisions. Au départ, $\mathcal{N}$ est vide.

Lorsque tous les polynômes de f sont constants en X_n, on calcule les tableaux de signes possibles de f pour la variable suivante (X_{n-1}). De ceux-ci on retient les colonnes de signes où les polynômes de f qui divisent un des polynômes de $\mathcal{N}$ sont effectivement non nuls.

Lorsqu'un des polynômes de f est non constant, on suit la procédure suivante :

soit c le premier coefficient dominant non constant de $f_1, \ldots, f_s$, qui ne divise pas un des polynômes de $\mathcal{N}$, i.e. qui peut être nul quand les polynômes de $\mathcal{N}$ sont non nuls; soit $f_i = cx^d + r$ le polynôme dont c est le coefficient dominant; on traite alors les deux cas :

- $\{f_1, \ldots, f_{i-1}, f_i, f_{i+1}, \ldots, f_s\}$ en ajoutant les facteurs sans carrés [2] de c à $\mathcal{N}$.

[2]si $P = \prod (h_i)^i$, avec les h_i non constants en X_n, premiers entre eux, de contenu 1 - i.e. leurs coefficients sont premiers entre eux- et sans facteurs carrés - i.e. premiers avec leur dérivée -, alors les facteurs sans carrés de P sont les h_i, plus les facteurs sans carrés de son contenu.

- $\{f_1, \ldots, f_{i-1}, r, f_{i+1}, \ldots, f_s, c\}$ dont on ne garde que les tableaux de signes ou c est identiquement nul.

soit il n'y a pas de tel c. Dans ce cas lorsque les polynômes de $\mathcal{N}$ sont non nuls, tous les coefficients dominants des polynômes de f sont non nuls, et on peut appliquer alors la méthode à une variable (dérivation et pseudo-divisions) car les hypothèses sont vérifiées.

2.3. Améliorations

On accélère considérablement l'algorithme en factorisant les polynômes avec la factorisation sans carrés qui a l'avantage d'être rapide à calculer (uniquement des pgcd de polynômes à plusieurs variables, que l'on calcule avec l'algorithme des sous-résultants [2]). Il est alors immédiat de calculer les tableaux des signes de f si on connaît ceux de la famille ses facteurs.

L'implémentation qu'on a faite en Ocaml traite les polynômes de $\mathbb{Z}[X_1, \ldots, X_n]$ [3]. En effet, dans ce cas, tous les calculs ne produisent que des polynômes dans cet anneau.

2.4. Un exemple

Soit $f = \{XY^3 + Y^2\}$, polynôme de $\mathbb{R}[X][Y]$. Au départ, $\mathcal{N} = \{\}$.

La factorisation sans carrés de f donne deux facteurs, qui constituent la nouvelle famille $f = \{XY + 1, Y\}$. Etudions ses coefficients dominants non constants :

$X \neq 0$: maintenant $\mathcal{N} = \{X\}$, et tous les coefficients dominants de f sont non nuls. Appliquons la méthode à une variable :

$f_1 = Y$, $f_2 = XY + 1$ puis $f_2' = X$, $g_1 = 0$, $g_2 = 1$ La factorisation sans carré élimine les polynômes constants: la famille courante f devient $\{Y, X\}$.
Les coefficients dominants sont tous non nuls : on peut dériver et on obtient la famille $\{X, 1, 0, 0\}$.
En factorisant, on est ramené à la famille $\{X\}$, constante en Y. On passe donc à la variable suivante, X. En dérivant on obtient $\{1, 0\}$, dont le tableau de signes est

1	+
0	0

D'où celui de $\{X\}$ qui est

X	$-$	0	+

Il y a donc trois tableaux de signes possibles pour $\{X\}$ considérée maintenant comme famille de polynômes (constants) en Y:

X	$-$

X	0

X	+

On n'en retient que les cas où X est non nul, puisque X divise un des polynômes de $\mathcal{N} = \{X\}$.
Restent donc deux tableaux:

X	$-$

X	+

En remontant à la famille $\{Y, X\}$, on a donc deux tableaux:

Y	$-$	0	+
X	$-$	$-$	$-$

Y	$-$	0	+
X	+	+	+

[3] On utilise la bibliothèque `Big_int` de Ocaml.

Puis à $\{XY+1, Y\}$, qui a deux tableaux de signes:

$XY+1$	$+$	$+$	$+$	0	$-$
Y	$-$	0	$+$	$+$	$+$

$XY+1$	$-$	0	$+$	$+$	$+$
Y	$-$	$-$	$-$	0	$+$

ce qui achève le cas $X \neq 0$.

$X = 0$: $\mathcal{N} = \{\}$ et la famille courante devient $\{1, Y\}$ puis $\{Y\}$ après la factorisation sans carrés. Après calcul pour la variable suivante X, on obtient un unique tableau de signes : $\{Y : [-, 0, +]\}$, qui est valide, puisque ses polynômes ne divisent aucun polynôme de $\mathcal{N}$. D'où un tableau de signes pour $\{XY+1, Y\}$ quand X est nul: $\{XY+1 : [+,+,+],\ Y : [-,0,+]\}$.

Finalement on a trois tableaux de signes pour la famille $\{XY+1, Y\}$:

$XY+1$	$+$	$+$	$+$	0	$-$
Y	$-$	0	$+$	$+$	$+$

$XY+1$	$-$	0	$+$	$+$	$+$
Y	$-$	$-$	$-$	0	$+$

$XY+1$	$+$	$+$	$+$
Y	$-$	0	$+$

D'où trois tableaux de signes possibles pour la famille de départ $\{XY^3+Y^2\}$:

XY^3+Y^2	$+$	0	$+$	0	$-$

XY^3+Y^2	$-$	0	$+$	0	$+$

XY^3+Y^2	$+$	0	$+$

correspondant aux cas $X < 0$, $X > 0$ et $X = 0$.

3. Trace des calculs.

Pour pouvoir construire la preuve qu'un ensemble de tableaux de signes correspond bien à ceux d'une famille de polynômes, on va utiliser une trace des calculs qui ont permis de produire ces tableaux de signes. Comme on l'a vu, l'algorithme fonctionne par étapes :

- on tranforme une famille f de polynômes en une famille f', soit par factorisation, soit par dérivation et divisions euclidiennes, soit en passant à la variable suivante,
- on calcule récursivement les tableaux de signes de f',
- à partir de ceux-ci on construit les tableaux de signes de f,
- l'arrêt a lieu lorsque tous les polynômes sont constants, donnant lieu à un tableau de signe trivial.

Une trace des calculs est donc une suite de traces élémentaires correspondant aux étapes de calculs. Chaque étape donne lieu éventuellement à des informations qui décrivent les résultats des calculs intermédiaires effectués, pour éviter de les refaire lors de la construction de la preuve :

Dérivation: on stocke dans la trace les paramètres c_i, q_i des pseudo-divisions $c_i f_s = q_i f_i + g_i$, $c_s f_s = q_s f'_s + g_s$.

Factorisation: on stocke les décompositions des polynômes de f en produits de polynômes de f' et d'un entier.

Une trace du calcul des signes d'une famille f en la variable v à partir de ceux de f' est finalement un des termes suivants :

- $Choix(f, v, l)$ où l est la liste de traces correspondant aux différents tableaux possibles de la famille f;
- $Derivation(f, v, w, I, t)$ où w est le tableau des signes de f, I une liste des paramètres des pseudo-divisions, et t est la trace de f';
- $Factorisation(f, v, w, I, t)$ où w est le tableau des signes de f, I une liste des factorisations de f en fonction de f', et t est la trace de f';
- $Cas(f, v, w, t)$ où w est le tableau des signes de f, et t est la trace de f'. f est constituée de constantes pour la variable v, w n'a qu'une colonne, prise parmi les tableaux possibles de f', qui est ici égale à f, pour la variable $v - 1$. La liste des traces de type *Cas* parcourt toutes les colonnes w possibles pour f';
- $Constantes(f, w)$ où w est le tableau des signes de f.

4. Elimination des quantificateurs.

La trace d'un calcul permet d'éliminer les quantificateurs universels dans une formule de la forme

$$\forall X_n, \ldots, X_{p+1},\ f_1 \#_1 0, \ldots, f_s \#_s 0$$

où $\#_i \in \{<, >, \leq, \geq, =\}$. En effet il suffit, pour chaque tableau de signes w de f compatible avec les équations ou inéquations $f_1 \#_1 0, \ldots, f_s \#_s 0$, de récupérer par un parcours en profondeur de la trace des calculs qui l'ont produit, les premiers tableaux de signes concernant la variable X_{p+1}.

Par exemple pour la formule $\forall X_4,\ X_1 X_4^2 + X_2 X_4 + X_3 > 0$, on obtient les tableaux de signes suivants:

X_1	0
X_3	+
X_2	0

X_1	+
$4X_1X_3 - X_2^2$	+

On a donc l'équivalence

$$(\forall X_4,\ X_1 X_4^2 + X_2 X_4 + X_3 > 0) \Leftrightarrow (X_1 = 0 \wedge X_3 > 0 \wedge X_2 = 0) \vee (X_1 > 0 \wedge 4X_1X_3 - X_2^2 > 0)$$

5. Construction des preuves pour Coq.

En Coq, les preuves sont des termes comme les autres. Dans ce langage typé, basé sur l'isomorphisme de Curry-Howard, le type d'une preuve p est le théorème t qu'elle démontre: on écrit alors $p : t$. La construction des preuves dans une déduction logique élémentaire est très simple, elle se fait par application d'une fonction à ses arguments. En effet, dans Coq, on identifie une implication logique

$A \rightarrow B$ à une fonction qui prend une preuve de A et qui rend une preuve de B. Ainsi, par exemple, si on a une preuve $H1$ de A, une preuve $H2$ de B, et une preuve H de $A \rightarrow B \rightarrow A \wedge B$, et bien $(H\ H1\ H2)$ est une preuve de $A \wedge B$. Toutes les preuves de Coq ne sont évidemment pas des applications [4] mais pour nos besoins ce sera suffisant: les preuves qu'on construira seront des applications successives de théorèmes d'analyse élémentaire.

Leur type en Ocaml est le suivant:

```
type preuve = Theo of string
     | Preuve of preuve list
;;
```

par exemple :

```
Preuve [Theo "(A,B:Prop)A/\B->A"; Theo "1>0/\0<1"]
```

est une preuve de `1>0` (on omet ici, comme en Coq, les arguments implicites).

La traduction de ces preuves en preuves Coq est immédiate.

5.1. Méthode.

Etant données une famille $f = \{f_1, \ldots, f_s\}$ de polynômes de $\mathbb{R}[X_1, \ldots, X_n]$, et des hypothèses de signes sur ces polynômes $H_1 : f_1 \#_1 0, \ \ldots, \ H_s : f_s \#_s 0$ où $\#_i \in \{<, >, \leq, \geq, =, \neq\}$ le but est de construire une contradiction, i.e. une preuve de la formule $\Phi = H_1 \wedge \ldots H_s \Rightarrow False$. Pour cela on utilisera une preuve de la formule :

$$\begin{array}{rl}\Psi = \forall x_1, \ldots, x_n, & \\ & \forall j, signe(f_j(x_1, \ldots, x_n)) = S_{1j} \\ \vee & \ldots \\ \vee & \forall j, signe(f_j(x_1, \ldots, x_n)) = S_{pj}\end{array}$$

où les signes S_{ij} sont tirés des tableaux de signes possibles de la famille f.

En effet il suffit de vérifier que les signes des hypothèses $H_1, \ldots, H_s$ sont incompatibles avec les familles de signes $S_{i1}, \ldots, S_{is}$.

On obtiendra une preuve de Ψ à partir des différents tableaux de signes de f. Soit w un tableau de signes de f pour la variable X_n. Sa signification est la formule :

$$\begin{array}{rl}\Psi_{fw} = & \exists x_1, \ldots, x_{n-1}, \exists z_1, \ldots, z_p, \\ & z_1 < z_2 \wedge \ldots \wedge z_{p-1} < z_p \\ \wedge & Signe_{w(I_1,1)}(f_1(x_1, \ldots, x_{n-1}, I_1)) \\ & \ldots \\ \wedge & Signe_{w(I_1,s)}(f_s(x_1, \ldots, x_{n-1}, I_1)) \\ \ldots & \\ \wedge & Signe_{w(I_{2p+1},1)}(f_1(x_1, \ldots, x_{n-1}, I_{2p+1})) \\ & \ldots \\ \wedge & Signe_{w(I_{2p+1},s)}(f_s(x_1, \ldots, x_{n-1}, I_{2p+1}))\end{array}$$

où $I_1, \ldots, I_{2p+1}$ est la subdivision de $]-\infty; +\infty[$ associée à $z_1, \ldots, z_p$, et les prédicats $Signe_+ = Pos$, $Signe_0 = Nul$, $Signe_- = Neg$ sont définis par $Pos(h, I) \Leftrightarrow \forall x \in I,\ h(x) > 0$, etc. Si on connaît $x_1, \ldots, x_{n-1}$ et $z_1, \ldots, z_p$, il suffit, pour obtenir une de preuve de Ψ_{fw}, d'avoir, pour chaque I_k et chaque i, une preuve de $Signe_{w(I_k,i)}(f_i(x_1, \ldots, x_{n-1}), I_k)$. Autrement dit un tableau de signes doit

[4] il y des abstractions, des points fixes, des traitement par cas,...

contenir pour chacun de ses signes, une preuve de ce signe : une case contient désormais un couple (signe, preuve).

Comme on l'a vu, l'algorithme de calcul des tableaux de signes procède par transformation successives de la famille f. On construit les tableaux de signes de f à partir de ceux de sa transformée f'. Pour les preuves des signes de ces tableaux, le procédé est analogue : on construira une preuve d'un signe $w(I,i)$ de f_i dans un intervalle I à partir de preuves de signes dans le tableau correspondant de la famille f'. Pour cela on utilisera soit le théorème des valeurs intermédiaires, soit des théorèmes élementaires sur le signe d'un polynôme en fonction de celui de ses facteurs, soit des théorèmes élémentaires sur les unions d'intervalles.

Pour obtenir les preuves d'existence des valeurs $x_1,\ldots,x_{n-1}$ et $z_1,\ldots,z_p$, on utilisera encore le théorème des valeurs intermédiaires et les valeurs et subdivision associées au tableau f'.

Enfin on construira, toujours à partir des preuves qui concernent f', la preuve que les tableaux de signes de f sont les seuls possibles.

On va maintenant décrire sur un exemple la construction des preuves dans le cas d'une variable. Le cas multi-variables n'est pas encore programmé, mais ne devrait pas poser de problème particulier.

5.2. Cas d'une variable, exemple.

Prenons l'exemple du polynôme X^2-1. L'algorithme donne la trace suivante :

```
Derivation
 ([|X^2-1|], 1,
  | + 0 - 0 + |,
  [1*(X^2-1) = X*X+(-1)],
  Factorisation
   ([|X; -1|], 1,
    | - 0 + |
    | - - - |,
    [X=1*X; -1=-1],
    Derivation
     ([|X|], 1,
      | - 0 + |,
      [1*X=X*1+0],
      Constantes ([|1; 0|],
                  | + |
                  | 0 |))))
```

Construisons maintenant les preuves pas à pas, en remontant dans la trace:

- les constantes 1 et 0: leur tableau de signe, avec les preuves:

  ```
  1 | (+,p1) |
  0 | (0,p2) |
  ```

  où

  ```
  p1 = Theo "(Pos 1 ]-inf;+inf[)"
  p2 = Theo "(Nul 0 ]-inf;+inf[)"
  ```

  en considérant que les théorèmes utilisés sont des théorèmes de base. On utilise à dessein dans les preuves une syntaxe proche de celle de Coq.

Avec ces preuves, la formule $\Psi_{fw} =$

$$Pos(1,]-\infty;+\infty[) \wedge Nul(0,]-\infty;+\infty[)$$

a pour preuve le terme

```
Preuve [Theo "(A1,A2:Prop)A1->A2->A1/\A2"; p1; p2]
```

- la famille $\{X\}$: son tableau de signes avec leurs preuves:

```
   X | (-,p3) (0,p4) (+,p5) |
```

avec

```
p3 = Preuve [and31;
             Preuve [Exists_prf;
                     Preuve [Preuve_val_interm1; Theo "(Derivee X 1)"; p1]]]
p4 = Preuve [and32;
             Preuve [Exists_prf;
                     Preuve [Preuve_val_interm1; Theo "(Derivee X 1)"; p1]]]
p5 = Preuve [and33;
             Preuve [Exists_prf;
                     Preuve [Preuve_val_interm1; Theo "(Derivee X 1)"; p1]]]
```

où

```
and31 = Theo "(A1,A2,A3:Prop)A1/\A2/\A3 -> A1"
and32 = Theo "(A1,A2,A3:Prop)A1/\A2/\A3 -> A2"
and33 = Theo "(A1,A2,A3:Prop)A1/\A2/\A3 -> A3"

theo_val_interm1 =
     Theo "(f,f':Pol)(Derivee f f') -> (Pos f' ]-inf;+inf[)
              -> (Exists x (Neg f ]-inf;x[)/\(Nul f [x])/\(Pos f ]x;+inf[))"
```

`Preuve_val_interm1` est une version particulière du théorème des valeurs intermédiaires,

si `p` est une de preuve de `Theo "(Exists x (P x))"`, alors `Preuve [Exists_val; p]` donne un `y` tel que `(P y)`, et `Preuve [Exists_prf; p]` est une preuve de `(P y)`.

Enfin, `(Derivee a b)` indique que b est la dérivée de a modulo un facteur strictement positif.

On a alors

$$\begin{array}{rl} \Psi_{fw} = & \exists z_1, \\ & Neg(X,]-\infty; z_1[) \\ \wedge & Nul(X, [z_1]) \\ \wedge & Pos(X,]z_1; +\infty[) \end{array}$$

dont une preuve est :

```
p6 = Preuve [theo_val_interm1; Theo "(Derivee X 1)"; p1]
```

- la famille $\{X, -1\}$: son tableau de signes est

```
   X | (-,p3) (0,p4) (+,p5) |
  -1 | (-,p7) (-,p8) (-,p9) |
```

où

```
z1= Preuve [Exists_val; p6]
p7 = Preuve [Theo "(x:R)(Neg f ]-inf;+inf[) -> (Neg f ]-inf;x[)";
             z1;
             Theo "(Neg -1 ]-inf;+inf[)"]
p8 = Preuve [Theo "(x:R)(Neg f ]-inf;+inf[) -> (Neg f [x])";
             z1;
             Theo "(Neg -1 ]-inf;+inf[)"]
p9 = Preuve [Theo "(x:R)(Neg f ]-inf;+inf[) -> (Neg f ]x;+inf[)";
             z1;
             Theo "(Neg -1 ]-inf;+inf[)"]
```

ainsi

$$\begin{array}{rrl}\Psi_{fw} = & & \exists z_1, \\ & & Neg(X,]-\infty; z_1[) \\ & \wedge & Nul(X, [z_1]) \\ & \wedge & Pos(X,]z_1; +\infty[) \\ & \wedge & Neg(-1,]-\infty; z_1[) \\ & \wedge & Neg(-1, [z_1]) \\ & \wedge & Neg(-1,]z_1; +\infty[) \end{array}$$

qui a pour preuve

```
Preuve [Exists_couple;
        z1;
        Preuve [Theo "(A1,A2,A3,A4,A5,A6:Prop)A1->...->A6->A1/\.../\A6";
                p3;p4;p5;p7;p8;p9]]
```

où `Exists_couple` construit une preuve de `(Exists x (P x))` à partir d'un `y` et d'une preuve de `(P y)`.

- enfin $X^2 - 1$: son tableau de signes avec les preuves :

```
  X^2-1 | (+,p10) (0,p11) (-,p12) (0,p13) (+,p14) |
```

avec

```
theo_val_interm2 =
   Theo "(f,f':Pol)(z:R)(Derivee f f') -> (Neg f' ]-inf;z[) -> Neg(f,[z])
        -> (Exists x (x<z)/\(Pos f ]-inf;x[)/\(Nul f [x])/\(Neg f ]x;z[))"
```

qui est une version du théorème des valeurs intermédiaires,

```
p15 = Preuve [Theo "(f,g,q,r,c:Pol)(x:R) c>0 -> Nul(g,x) -> Neg(r,x) ->
                    -> c*f=q*g+r -> Neg(f,x)";
              z1; Theo "1>0"; p4; p8; Theo "1*(X^2-1)=X*X-1"]
```

qui est une preuve que $X^2 - 1$ est négatif en `z1`,

```
p20 = Preuve [theo_val_interm2;
              z1;
              Theo "(Derivee X^2-1 X)";
```

```
              p3;
              p15]
p10 = Preuve [and42; Preuve [Exists_prf; p20]]

p11 = Preuve [and43; Preuve [Exists_prf; p20]]

p16 = Preuve [and44; Preuve [Exists_prf; p20]]
```

En appliquant de nouveau la version suivante du théorème des valeurs intermédiaires, on obtient les autres preuves du tableau :

```
theo_val_interm3 =
   Theo "(f,f':Pol)(z:R)(Derivee f f') -> (Pos f' ]z;+inf[) -> Neg(f,[z])
        -> (Exists x (z<x)/\(Neg f ]z;x[)/\(Nul f [x])/\(Pos f ]x;+inf[))"

p21 = Preuve [theo_val_interm3;
              z1;
              Theo "(Derivee X^2-1 X)";
              p5;
              p15]
p17 = Preuve [and42; Preuve [Exists_prf; p21]]
p13 = Preuve [and43; Preuve [Exists_prf; p21]]
p14 = Preuve [and44; Preuve [Exists_prf; p21]]
```

et enfin,

```
p12 = Preuve [Theo "(f:Pol)(x,y,z:R)x<y -> y<z ->
                     Neg(f,]x;y[)->Neg(f,[y])->Neg(f,]y;z[)
                     -> Neg(f,]x;z[)";
              Preuve [and41; Preuve [Exists_prf; p20]];
              Preuve [and41; Preuve [Exists_prf; p21]];
              p16;
              p15;
              p17]
```

finalement

$$
\begin{array}{rcl}
\Psi_{fw} = & & \exists z_2, z_3 \\
& & z_2 < z_3 \\
& \wedge & Pos(X^2-1,]-\infty; z_2[) \\
& \wedge & Nul(X^2-1, [z_2]) \\
& \wedge & Neg(X^2-1,]z_2; z_3[) \\
& \wedge & Nul(X^2-1, [z_3]) \\
& \wedge & Pos(X^2-1,]z_3; +\infty[)
\end{array}
$$

avec sa preuve

```
Preuve [Exists_couple;
        z2;
        Preuve [Exists_couple;
                z3;
                Preuve [Theo "(A1,A2,A3,A4,A5,A6:Prop)A1->...->A6->A1/\.../\A6";
                        Preuve [Theo "(x,y,z:R)x<y->y<z->x<z";
                                Preuve [and41; Preuve [Exists_prf; p20]];
```

```
                                Preuve [and41; Preuve [Exists_prf; p21]]];
                        p10;p11;p12;p13;p14]]]
```

où

```
z2 = Preuve [Exists_val; p20]
z3 = Preuve [Exists_val; p21]
```

6. Travail futur.

A l'heure où cet article est écrit, les choses en sont au stade suivant :

- L'algorithme d'élimination des quantificateurs est écrit en Ocaml et fonctionne sur des exemples simples mais non triviaux, comme l'équation du troisième degré (voir en annexe). Il fonctionne aussi sur des exemples tirés des preuves sur les réels de la bibliothèque de Coq. Enfin il fonctionne sur l'équation du quatrième degré, mais échoue à reconstruire la trace complète des calculs par manque de place mémoire ($>$1 Giga). Un tel algorithme général ne produit en effet pas sur cette entrée le calcul optimal qu'on sait effectuer[4] pour ce problème.
- La construction de la preuve Coq dans le cas d'une variable est à 90% effectuée. Reste à prouver en Coq les théorèmes sur les réels utilisés.
- Un prototype de tactique, appelée «Tarski» fonctionne en Coq : cette tactique appelle l'algorithme en Ocaml, et, s'il répond par un succès, prouve sauvagement le but à l'aide d'un axiome `(P:Prop)P`. Evidemment le but est de vérifier la faisabilité de l'intégration du code caml avec Coq, et aussi de pouvoir tester facilement et confortablement des exemples variés de preuves sur les réels.

L'extension de la construction de la preuve Coq au cas de plusieurs variables devrait demander beaucoup moins de travail que ce qui a été nécessaire au cas d'une variable, où les nombreux cas d'application du théorème des valeurs intermédiaires sont assez fastidieux.

Les auteurs tiennent à remercier les relecteurs pour les remarques et indications bibliographiques qu'ils nous ont fournies, ainsi que Renaud Rioboo pour le «bug» qu'il a soulevé dans l'équation de degré 3.

Bibliographie

[1] J. Bochniak, M. Coste, M-F. Roy : *Géométrie Algébrique Réelle*, Springer-Verlag (1986), pages 15-19.

[2] S. Boulmé: *Vers la spécification formelle d'une preuve d'un algorithme non trivial de calcul formel: le calcul du pgcd par la chaîne des pseudo-restes des sous-résultants*, Stage de DEA, LIP6, sept 96.

[3] The Coq Developpement Team: *The coq proof assistant, reference manual*, `http://pauillac.inria.fr/coq/doc/main.html`, 2001.

[4] D. Lazard: *Quantifier Elimination: Optimal Solution for Two Classical Examples*, J. Symbolic Computation (1988) **5**, 261-266.

[5] M. Hirschowitz, M. Chiaverini: *Preuve en Coq du théorème des valeurs intermédiaires*, projet de maîtrise MIM, UNSA, printemps 2000.

[6] L. Hörmander: *The analysis of linear partial differental operators, vol.2*, Springer-Verlag (1986).

[7] X. Leroy et al: *The objective caml system*, *INRIA*, `http://caml.inria.fr/ocaml/htmlman/`, 2001.

[8] A. Mahboubi: *Programmation d'une tactique dans le système Coq pour la méthode de Tarski-Seidenberg*, rapport de stage de deuxième année
du Magistère de Mathématiques et Applications de l'ENS Lyon, septembre 2001.

Annexe

On donne ici le résultat de l'élimination de x dans $x_1x^3 + x_2x^2 + x_3x + x_4$. Chaque implication donne les signes de polynômes en x_1, x_2, x_3 et x_4 en condition, et en conclusion le tableau de variation de $x_1x^3 + x_2x^2 + x_3x + x_4$ entre ∞ et $+\infty$. On remarquera que certaines implications peuvent être simplifiées et regroupées, ce que l'algorithme ne fait pas encore.

```
[| - | x1
 | + | 3*x1*x3+(-1)*x2^2
 | + | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
 | - | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
  ==> | + 0 - | x1*x5^3+x2*x5^2+x3*x5+x4
      ;
 | - | x1
 | + | 3*x1*x3+(-1)*x2^2
 | + | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
 | 0 | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
  ==> | + 0 - | x1*x5^3+x2*x5^2+x3*x5+x4
      ;
 | - | x1
 | + | 3*x1*x3+(-1)*x2^2
 | + | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
 | + | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
  ==> | + 0 - | x1*x5^3+x2*x5^2+x3*x5+x4
      ;
 | - | x1
 | - | 3*x1*x3+(-1)*x2^2
 | + | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
 | - | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
  ==> | + 0 - | x1*x5^3+x2*x5^2+x3*x5+x4
      ;
 | - | x1
 | - | 3*x1*x3+(-1)*x2^2
 | 0 | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
 | - | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
  ==> | + 0 - 0 - | x1*x5^3+x2*x5^2+x3*x5+x4
      ;
 | - | x1
 | - | 3*x1*x3+(-1)*x2^2
 | - | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
 | - | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
  ==> | + 0 - 0 + 0 - | x1*x5^3+x2*x5^2+x3*x5+x4
      ;
 | - | x1
 | - | 3*x1*x3+(-1)*x2^2
 | - | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
 | 0 | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
  ==> | + 0 - 0 + 0 - | x1*x5^3+x2*x5^2+x3*x5+x4
      ;
 | - | x1
 | - | 3*x1*x3+(-1)*x2^2
 | - | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
 | + | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
  ==> | + 0 - 0 + 0 - | x1*x5^3+x2*x5^2+x3*x5+x4
      ;
 | - | x1
 | - | 3*x1*x3+(-1)*x2^2
 | 0 | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
 | + | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
  ==> | + 0 + 0 - | x1*x5^3+x2*x5^2+x3*x5+x4
      ;
 | - | x1
 | - | 3*x1*x3+(-1)*x2^2
 | + | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
 | + | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
  ==> | + 0 - | x1*x5^3+x2*x5^2+x3*x5+x4
      ;
 | + | x1
 | - | 3*x1*x3+(-1)*x2^2
 | + | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
 | - | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
  ==> | - 0 + | x1*x5^3+x2*x5^2+x3*x5+x4
      ;
 | + | x1
 | - | 3*x1*x3+(-1)*x2^2
 | 0 | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
 | - | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
  ==> | - 0 - 0 + | x1*x5^3+x2*x5^2+x3*x5+x4
      ;
 | + | x1
 | - | 3*x1*x3+(-1)*x2^2
 | - | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
 | - | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
  ==> | - 0 + 0 - 0 + | x1*x5^3+x2*x5^2+x3*x5+x4
      ;
 | + | x1
 | - | 3*x1*x3+(-1)*x2^2
 | - | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
 | 0 | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
  ==> | - 0 + 0 - 0 + | x1*x5^3+x2*x5^2+x3*x5+x4
      ;
 | + | x1
 | - | 3*x1*x3+(-1)*x2^2
 | - | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
 | + | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
  ==> | - 0 + 0 - 0 + | x1*x5^3+x2*x5^2+x3*x5+x4
```

```
     ;
| + | x1
| - | 3*x1*x3+(-1)*x2^2
| 0 | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
| + | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
 ==> | - 0 + 0 + | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| + | x1
| - | 3*x1*x3+(-1)*x2^2
| + | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
| + | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
 ==> | - 0 + | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| + | x1
| + | 3*x1*x3+(-1)*x2^2
| + | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
| - | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
 ==> | - 0 + | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| + | x1
| + | 3*x1*x3+(-1)*x2^2
| + | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
| 0 | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
 ==> | - 0 + | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| + | x1
| + | 3*x1*x3+(-1)*x2^2
| + | 27*x1^2*x4^2+((-18)*x1*x2*x3+4*x2^3)*x4+4*x1*x3^3+(-1)*x2^2*x3^2
| + | 27*x1^2*x4+(-9)*x1*x2*x3+2*x2^3
 ==> | - 0 + | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| - | x1
| 0 | 3*x1*x3+(-1)*x2^2
| + | 9*x1*x4+(-1)*x2*x3
 ==> | + 0 - | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| - | x1
| 0 | 3*x1*x3+(-1)*x2^2
| 0 | 9*x1*x4+(-1)*x2*x3
 ==> | + 0 - | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| - | x1
| 0 | 3*x1*x3+(-1)*x2^2
| - | 9*x1*x4+(-1)*x2*x3
 ==> | + 0 - | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| + | x1
| 0 | 3*x1*x3+(-1)*x2^2
| - | 9*x1*x4+(-1)*x2*x3
 ==> | - 0 + | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| + | x1
| 0 | 3*x1*x3+(-1)*x2^2
| 0 | 9*x1*x4+(-1)*x2*x3
 ==> | - 0 + | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| + | x1
| 0 | 3*x1*x3+(-1)*x2^2
| + | 9*x1*x4+(-1)*x2*x3
 ==> | - 0 + | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| - | x2
| 0 | x1
| + | 4*x2*x4+(-1)*x3^2
 ==> | - - - | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| - | x2
| 0 | x1
| 0 | 4*x2*x4+(-1)*x3^2
 ==> | - 0 - | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| - | x2
| 0 | x1
| - | 4*x2*x4+(-1)*x3^2
 ==> | - 0 + + + 0 - | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| + | x2
| 0 | x1
| - | 4*x2*x4+(-1)*x3^2
 ==> | + 0 - - - 0 + | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| + | x2
| 0 | x1
| 0 | 4*x2*x4+(-1)*x3^2
 ==> | + 0 + | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| + | x2
| 0 | x1
| + | 4*x2*x4+(-1)*x3^2
 ==> | + + + | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| 0 | x1
| 0 | x2
| - | x3
 ==> | + 0 - | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| 0 | x1
| 0 | x2
| + | x3
 ==> | - 0 + | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| 0 | x1
```

```
| 0 | x2
| - | x4
| 0 | x3
 ==> | - | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| 0 | x1
| 0 | x2
| 0 | x4
| 0 | x3
 ==> | 0 | x1*x5^3+x2*x5^2+x3*x5+x4
     ;
| 0 | x1
| 0 | x2
| + | x4
| 0 | x3
 ==> | + | x1*x5^3+x2*x5^2+x3*x5+x4
     ]
```