Décomposition Algébrique Cylindrique en Coq/Rocq

Quentin Vermande

▶ **To cite this version:**

Quentin Vermande. Décomposition Algébrique Cylindrique en Coq/Rocq. 36es Journées Francophones des Langages Applicatifs (JFLA 2025), Jan 2025, Roiffé, France. hal-04859512

HAL Id: hal-04859512
https://inria.hal.science/hal-04859512v1

Submitted on 30 Dec 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Copyright

Décomposition Algébrique Cylindrique en Coq/Rocq

Quentin Vermande

Université Côte d'Azur, INRIA

La Décomposition Algébrique Cylindrique (ou CAD) est un outil fondamental de la géométrie semi-algébrique. C'est un algorithme doublement exponentiel dont l'application phare est l'élimination des quantificateurs dans les formules de la théorie des corps réels clos. En particulier, il permet de décider si des systèmes d'inéquations polynomiales sont satisfiables. Cet article décrit la première formalisation de la correction de cet algorithme dans un assistant de preuve. Si l'algorithme n'est pas implémenté de façon à être exécuté, il s'agit d'une étape vers une implémentation vérifiée de la CAD, où les seuls éléments manquants sont des implémentations efficaces des structures de données et des algorithmes requis.

1 Introduction

Une Décomposition Algébrique Cylindrique (ou CAD) est une partition de l'espace $\mathbb{R}^n$ en cellules décrites par des systèmes d'inéquations polynomiales et telles que les projections sur $\mathbb{R}^m$ pour $m < n$ produisent encore des CAD. Cette dernière condition de compatibilité justifie l'usage du mot « cellule » pour les éléments de la CAD, au sens où projeter une cellule sur l'espace obtenu en ne gardant que les premières coordonnées produit encore une cellule. Il s'agit des même cellules qu'en géométrie o-minimale.

Les décompositions qui nous intéressent sont celles qui sont adaptées à une famille finie $\mathcal{P}$ de polynômes à n variables, au sens où chaque polynôme de $\mathcal{P}$ est de signe constant sur chaque cellule. Cette notion a été introduite par Collins [Col75] en 1975 dans le but de décider rapidement de la satisfiabilité de formules de la théorie des corps réels clos. Il a été le premier à obtenir un algorithme doublement exponentiel en le nombre de variables et polynomial en les autres paramètres.

Depuis son introduction, il y a eu beaucoup d'optimisations de l'algorithme de calcul de la CAD, qui a eu un large champ d'applications incluant des problèmes de robotique [SS83], des problèmes d'optimisation [HJX16] et la vérification de systèmes dynamiques [ST11] et de systèmes hybrides [Pla22]. En tant qu'outil fondamental de la géométrie semi-algébrique, il est omniprésent dans les systèmes de calcul formel. Cet algorithme est connu pour la complexité de sa preuve de correction et a par conséquent été évité dans d'autres travaux de formalisation, comme l'implémentation de l'élimination des quantificateurs en Isabelle/HOL et PVS [KTP23, MND18]. Par exemple, la formalisation en Isabelle/HOL utilise une variante de l'algorithme de Ben-Or, Kozen et Reif [BOKR86], qui est plus simple et moins efficace. Nous n'avons pas connaissances de travaux en Lean sur ce sujet. Un algorithme de calcul de la CAD a déjà été implémenté en Coq/Rocq [Mah05] mais sans preuve de correction.

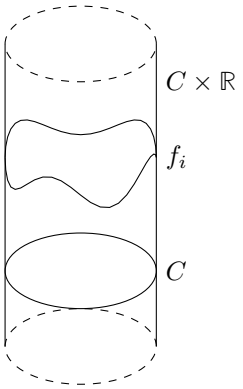
Nous présentons ici une formalisation de la version originelle de l'algorithme par Collins dans l'assistant de preuve Coq/Rocq et une preuve formalisée de sa correction. La formalisation se trouve dans le projet github correspondant, à l'adresse `https://github.com/math-comp/cad/tree/v1.1`. Nous suivons à quelques détails près la présentation donnée dans [BPR06], les points de divergence seront détaillés au cours de l'exposition. Notamment, la preuve de la Proposition 5.14 n'est pas donnée dans le livre et a requis une adaptation de celle de la Proposition 5.13, et non pas une preuve par une récurrence triviale comme le livre le suggère.

L'implémentation n'est pas faite pour être exécutée, notamment car elle utilise des structures de données inefficaces. De plus, la procédure d'isolation des racines que nous utilisons ne calcule pas. Pour obtenir un programme exécutable, il faudrait implémenter des structures de données et des algorithmes efficaces et prouver leur équivalence avec ceux que nous utilisons. Nous utilisons la bibliothèque *MathComp*, et plus précisément son extension à la géométrie semi-algébrique [MC12, Dja18]. Certaines des primitives que nous utilisons ont déjà été formalisées, comme l'isolation des racines d'un polynôme de $\mathbb{R}[X]$ [MC12], tandis que d'autres, comme l'obtention de points entre deux racines d'un polynôme, ne le sont pas. Le travail de formalisation inclut un certain nombre de résultats qui ne seront pas détaillés ici, comme l'obtention du nombre de racines réelles distinctes d'un polynôme par des calculs de sous-résultants [BPR06] (dans `subresultant.v` dans notre formalisation), le théorème de continuité des racines d'un polynôme [NR24] (dans `continuity_roots.v`), des résultats originaux de semi-algébraicité de fonctions (dans `semialgebraic.v`), le théorème de la borne supérieure pour les ensembles semi-algébriques [BPR06] (dans `semialgebraic.v`), la notion de connexité semi-algébrique [BPR06] (dans `topology.v`) et quelques résultats classiques sur des transformations de polynômes multivariés (dans `auxresults.v`).

Nous prouvons deux algorithmes, l'Algorithme 1 qui calcule les cellules d'une CAD adaptée à une famille de polynômes et l'Algorithme 2 qui calcule un échantillon d'une telle CAD, c'est-à-dire un ensemble de points contenant un point par cellule d'une telle décomposition. Ce deuxième algorithme permet de vérifier efficacement s'il existe un point où une famile de polynômes satisfait une condition de signe donnée.

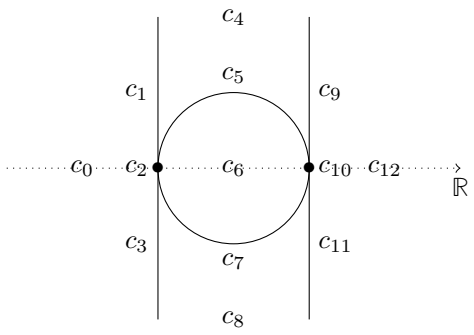
Description haut niveau de l'algorithme

En quelques mots, l'algorithme fonctionne par récurrence sur la dimension n, le cas de base $n = 0$ étant trivial. En dimension $n+1$, étant donnée une famille finie de polynômes $\mathcal{P} \subset \mathbb{R}[X_0, ..., X_n]$, nous calculons une famille finie $\mathcal{Q}$ de polynômes à n variables et nous obtenons récursivement une CAD pour $\mathcal{Q}$. Puis, pour chaque cellule C de cette décomposition, nous décrivons, pour $(x_0, ..., x_{n-1}) \in C$, les racines collectives des $P(x_0, ..., x_{n-1}, X)$, $P \in \mathcal{P}$ par une séquence $f_1, ..., f_d : C \to \mathbb{R}$ de fonctions strictement croissantes continues. Nous obtenons alors des cellules en dimension $n + 1$ en partitionnant le cylindre $C \times \mathbb{R}$ le long des graphes de $f_1, ..., f_d$. Découper un cylindre le long du graphe d'une telle fonction produit 3 cellules, le graphe de la fonction et les parties du cylindre au dessus et en dessous du graphe, de sorte qu'on construit ainsi $2d + 1$ cellules.



Exemple 1. Pour $\mathcal{P} = \{X^2 + Y^2 - 1\}$, nous obtenons la décomposition suivante avec 13 cellules, où $X^2 + Y^2 - 1$ est strictement positif en dehors du disque unité fermé (i.e. sur les cellules $c_0, c_1, c_3, c_4, c_9, c_9, c_{11}$ et c_{12}), nul sur le cercle unité (i.e. sur les cellules c_2, c_5, c_7 et $c_1 0$) et strictement négatif dans le disque unité ouvert (i.e. sur la cellule c_6). Pour s'assurer qu'il s'agit bien d'une CAD, il faut vérifier que chaque cellule est décrite par des inéquations polynomiales. Par exemple, l'arc de cercle au dessus de l'axe des abscisses est décrit par les équations $Y > 0$ et $X^2 + Y^2 - 1 = 0$. Par ailleurs, les projections sur $\mathbb{R}^0$ et $\mathbb{R}^1$ sont

respectivement $\{0\}$ et la droite réelle coupée en -1 et 1 qui sont bien des CAD. Le polynôme $X^2 + Y^2 - 1$ est de signe constant sur chaque cellule, donc la décomposition ci-dessous est une CAD adaptée à $X^2 + Y^2 - 1$.



La section 2 décrit les algorithmes dont nous prouvons la correction dans la section 3. Certaines preuves ne seront pas retranscrites, toutes sont consultables dans le projet github [1] concerné.

Conventions et prérequis Dans tout le reste de cet article, nous considérons R un corps réel clos et nous appelons $R[i]$ le corps des complexes sur R, vu que $R[i]$ est à R ce que $\mathbb{C}$ est à $\mathbb{R}$. Un corps réel clos est un corps muni d'un ordre compatible avec l'addition et la multiplication du corps et tel que $R[i]$ est algébriquement clos. L'exemple usuel est le corps $\mathbb{R}$ des nombres réels (auquel cas $\mathbb{R}[i] = \mathbb{C}$), mais nous pouvons aussi citer le corps $\mathbb{R}_{\text{alg}}$ des nombres réels algébriques. Ce niveau de généralité a un coût, car les corps réels clos ne sont pas nécessairement connexes, ce qui pose des problèmes pour les arguments de topologie que l'on souhaite utiliser. Par exemple, $\mathbb{R}_{\text{alg}}$ ne contient pas π, donc $]-\infty, \pi[\cup]\pi, +\infty[$ est un recouvrement de $\mathbb{R}_{\text{alg}}$ par des ouverts disjoints [BPR06].

Notre domaine d'étude est la géométrie semi-algébrique, qui est la géométrie des parties de R^n (où $n \in \mathbb{N}$) sur lesquelles des polynômes à n variables prennent certains signes. En première approximation, le lecteur peut remplacer toutes les occurrences de « semi-algébrique » par « défini par un système d'inégalités polynomiales ».

Nous notons deg'(P) le degré d'un polynôme, avec la convention de *MathComp* selon laquelle deg'$(0) = 0$, de sorte que deg'(0) soit un entier naturel.

2 Algorithmes

Nous définissons tout d'abord l'objet que l'algorithme de calcul de la CAD produit. Une décomposition cylindrique algébrique (ou CAD) est définie par récurrence sur la dimension de l'espace ambiant. Cette définition repose sur la notion de fonction semi-algébrique. Nous disons qu'une partie S de R^n est *semi-algébrique* lorsque S est l'ensemble des points de R^n satisfaisant une formule de la logique du premier ordre donnée sur le langage des anneaux ordonnés. Ensuite, une fonction de R^n dans R^m est semi-algébrique lorsque son graphe est semi-algébrique. Les cellules en dimension $n+1$ sont obtenues en découpant un cylindre $C \times R$ le long de graphes de fonctions $f_1, ..., f_d : C \to R$. Pour $f : C \to R$, nous notons `partition_of_graphs`(C, f) l'ensemble des cellules obtenues ainsi à partir de C et f.

Définition 1 (`isCylindricalDecomposition` dans la formalisation)**.** Une partition finie S de R^n est une *décomposition cylindrique algébrique* lorsque $n = 0$ ou, avec $\pi : R^n \to R^{n-1}$ la projection canonique et $T = \{\pi(C), C \in S\}$, T est une CAD de R^{n-1} et, pour tout $C \in T$, il existe une suite $f_1, ..., f_d$ strictement croissant de fonctions semi-algébriques continues

1. Identifiant Software Heritage : `swh:1:dir:026ed0a7c6c77d29872c1c8edb2d8932ea973211`

de R^{n-1} dans R^1 telles que les éléments de S qui se projettent sur C soient les cellules de `partition_of_graphs`(C, f).

En Coq/Rocq et avec quelques simplifications mineures, cela donne la définition ci-dessous. Comme dans la définition mathématique donnée ci-dessus, on spécifie les CAD par récurrence sur la dimension n. La contrainte commune à toutes les dimensions est que S doit être une partition de R^n. Puis, si $n = 0$ il n'y a pas de contrainte supplémentaire, sinon on construit S' en projetant tous les éléments de S sur R^{n-1}. On s'assure récursivement que S' est une CAD en dimension $n-1$ et que, pour toute cellule s' de S', il existe une liste finie de fonctions semi-algébriques continues sur s', ordonnée selon l'ordre point à point des fonctions et telle que les cellules de S qui se projettent sur s' sont exactement les cellules obtenues par le découpage du cylindre $S \times R$ avec les fonctions précédentes.

```
Fixpoint isCylindricalDecomposition n (S : {fset {SAset R^n}}) :=
  SAset_partition S
  /\ match n with | 0 => True | n.+1 =>
    let S' := [fset (s : {SAset R^n}) | s in S] in
    isCylindricalDecomposition S'
    /\ forall (s' : S'),
    exists m, exists xi : m.-tuple {SAfun R^n -> R^1},
      (forall i, {within [set` val s'], continuous (tnth xi i)})
      /\ sorted <%SA xi
      /\ [fset s in S | (s : {SAset R^n}) == val s']
         = partition_of_graphs (val s') xi
  end.
```

Exemple 2. Justifions que la partition $S = \{c_0, ..., c_{12}\}$ de l'exemple 1 est bien une CAD. On voit déjà que c'est une partition de R^2. Ensuite, on pose $S' = \{]-\infty, -1[, \{-1\},]-1, 1[, \{1\},]1, +\infty[\}$, obtenu en projetant les cellules de S sur R. Admettons qu'il s'agit bien d'une CAD. Considérons une cellule s' de S', par exemple $s' =]-1, 1[$. Les arcs du cercle de centre 0 et de rayon 1 sont les graphes des fonctions $f_0 : x \mapsto -\sqrt{1-x^2}$ et $f_1 : x \mapsto \sqrt{1-x^2}$, définies sur $s' =]-1, 1[$. Les graphes de f_0 et f_1 sont définies par les formules $-1 < X < 1 \wedge Y^2 + X^2 = 1 \wedge Y < 0$ et $-1 < X < 1 \wedge Y^2 + X^2 = 1 \wedge 0 < Y$, donc f_0 et f_1 sont semi-algébriques. Posons donc $m = 2$ et `xi` $= (f_0, f_1)$. Les fonctions f_0 et f_1 sont bien continues et $f_0 < f_1$, de sorte que la liste `xi` est bien triée. Enfin, les cellules de C qui se projettent sur s', qui sont c_4, c_5, c_6, c_7 et c_8, sont bien les cellules obtenues en découpant le cylindre $s' \times R$ le long des graphes de f_0 et f_1. On traite de la même manière les autres cellules de S'.

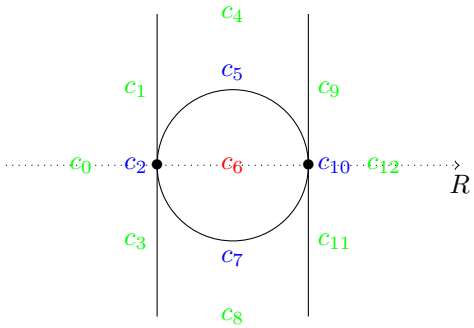
Le fait que la décomposition vient d'un découpage de cylindres se matérialise par les 4 demi-droites qui a priori n'ont aucune raison de faire partie de la décomposition, puisqu'elles ne séparent pas deux cellules sur lesquelles $X^2 + Y^2 - 1$ prend des signes différents. Les cylindres $]-\infty, -1[\times R$ et $]1, +\infty[\times R$ ne sont pas découpés, les droites $\{-1\} \times R$ et $\{1\} \times R$ sont découpées en 0 et le cylindre central $]-1, 1[\times R$ est découpé le long des deux fonctions dont l'union des graphes définit le cercle unité.

Il y a deux différences mineures avec la définition donnée dans [BPR06]. Cette dernière commence avec la dimension 1, mais commencer avec la dimension 0 ne change pas les cas récursif des algorithmes et rend les initialisations triviales. De plus, iels définissent une CAD comme une séquence $S_1, ..., S_n$ de partitions, tandis que nous n'introduisons que S_n, puisque les autres s'en déduisent par projection. Les décompositions qui nous intéressent sont celles qui respectent une famille de polynômes, au sens suivant :

Définition 2 (`poly_adapted`)**.** Soit $\mathcal{P}$ une famille de polynômes à n variables. Une partition S de R^n est *adaptée* à $\mathcal{P}$ lorsque, pour tous $C \in S$ et $P \in \mathcal{P}$, P a un signe constant sur C.

Exemple 3. Dans l'exemple 1, la partition est bien adaptée à $X^2 + Y^2 - 1$. En effet, $X^2 + Y^2 - 1$ prend un signe négatif sur la boule unité ouverte, qui est la cellule centrale

c_6 de la décomposition, s'annule sur les quatre cellules c_2, c_5, c_7 et c_{10} dont l'union est le cercle unité, et a un signe positif sur toutes les autres cellules.



Les fonctions utilisées pour faire le découpage des cellules sont obtenues en calculant des racines de polynômes. Cependant, en dimension n nous manipulons des polynômes à n variables. Nous définissons donc l'évaluation partielle, qui permet d'évaluer un tel polynôme en ses $n-1$ premières variables.

Définition 3 (`evalpmp`)**.** Soit $P \in R[X_0, ..., X_n]$. L'*évaluation partielle* de P est la fonction $\vec{P} : R^n \to R[X]$ définie par :

$$\forall x \in R^n, \vec{P}(x) = P(x_0, ..., x_{n-1}, X)$$

Exemple 4. L'évaluation partielle garde la dernière variable et évalue toutes les autres. Pour $P = X^2 + Y^2 - 1$, cela donne la fonction $\vec{P} : x \mapsto Y^2 + (x^2 - 1)$. Quand $|x| < 1$, $\vec{P}(x)$ a exactement deux racines, qui sont $\pm\sqrt{1 - x^2}$, quand $x = 1$, $\vec{P}(x) = Y$ a exactement une racine et quand $|x| > 1$, $\vec{P}(x)$ n'a pas de racine. Dans l'exemple 1, c'est ainsi que l'on décide où découper les cylindres pour obtenir les cellules finales.

Étant donné un ensemble fini $\mathcal{P}$ de polynômes à n variables, nous décrivons deux algorithmes. L'Algorithme 1 calcule une CAD adaptée à $\mathcal{P}$ et l'Algorithme 2 calcule une partie finie S de R^n telle qu'il existe une CAD adaptée à $\mathcal{P}$ dont toutes les cellules contiennent au moins un point de S. Ces algorithmes fonctionnent par récurrence sur la dimension de l'espace ambiant. Chaque étape récursive contient une phase de projection, où l'on calcule un ensemble fini de polynômes à $n-1$ variables et une phase de remontée, où l'on utilise le résultat du calcul récursif sur cet ensemble de polynômes pour obtenir le résultat final. Décrire précisément la phase de projection requiert l'introduction des sous-résultants et de leurs propriétés [BPR06], ce qui sort du cadre de cet article. Par conséquent nous noterons elim l'opération sous-jacente, appelée *élimination*, et donnerons sans preuve les conséquences de la correction de l'algorithme de calcul de la CAD sur elim($\mathcal{P}$).

Nous ne donnons pas de code pour l'Algorithme 1, qui est sous-jacent dans sa preuve de correction. Pour des raisons de lisibilité, le code pour l'Algorithme 2 est contenu dans deux fonctions, la première évaluant partiellement une famille de polynômes en un point donné puis calcule les racines collectives de ces polynômes et l'échantillon associé et la deuxième agrégeant ces calculs pour produire le résultat final. Plus précisément, la fonction auxiliaire produit les éléments l'échantillon au dessus d'un point s en ajoutant une coordonnée à s. Les vecteurs de R^n étant représentés par des matrices lignes, cela est fait en concaténant 's' avec le matrice à 1 ligne et 1 colonne contenant la coordonnée supplémentaire, ce qui donne une matrice à 1 ligne et `n+1` colonnes, puis en utilisant la fonction `castmx` pour transformer cette matrice en matrice à 1 ligne et `n.+1` colonnes. Cette opération est nécessaire car le vérificateur de types de Coq/Rocq ne peut pas identifier `n+1` et `n.+1`.

Input : $n \in \mathbb{N}$ et une partie fini $\mathcal{P}$ de $R[X_0, ..., X_{n-1}]$
Output : Une CAD S adaptée à $\mathcal{P}$

if $n = 0$ **then**
  $S = \{R^0\}$
else
  Soit $T = \text{CAD}(\text{elim}(\mathcal{P}))$
  $S = \varnothing$
  for $C \in T$ **do**
    Soit $x \in C$
    Soit $\mathcal{P}_* = \{P \in \mathcal{P}, \vec{P}(x) \neq 0\}$
    Soit $Q = \prod_{P \in \mathcal{P}_*} \vec{P}$
    Soit $d = |Q^{-1}(\{0\})|$
    Soit, pour $i \in [\![0, d-1]\!]$, $f_i : R^{n-1} \to R$ qui envoie $y \in C$ sur la i-ième racine de $Q(y)$ et $y \in R^{n-1} \setminus C$ sur i
    $S = S \cup \texttt{partition\_of\_graphs}(C, f)$
  end
end
return S

Algorithme 1 : Algorithme CAD

```
Definition lift_sample_cylindrical_decomposition n
                (P : {fset {mpoly R[n.+1]}}) (s : 'rV[R]_n) :=
  [fset castmx (erefl, (@addn1 n)) (row_mx s (\row__ x)) | x in
      let r := rootsR
        (\prod_(p : P | evalpmp s (muni (val p)) != 0)
            evalpmp s (muni (val p))) in
      (head 0 r - 1) :: (last 0 r + 1) :: r
      ++ [seq (r`_i.+1 + r`_i) / 2 | i <- iota 0 (size r).-1]
    ].

Fixpoint sample_cylindrical_decomposition n :
    {fset {mpoly R[n]}} -> {fset 'rV[R]_n} :=
  match n with
  | 0 => fun=> [fset \row__ 0]
  | S n => fun P =>
      let S := sample_cylindrical_decomposition (elimp P) in
      \big[fsetU/fset0]_(s : S)
        lift_sample_cylindrical_decomposition P (val s)
  end.
```

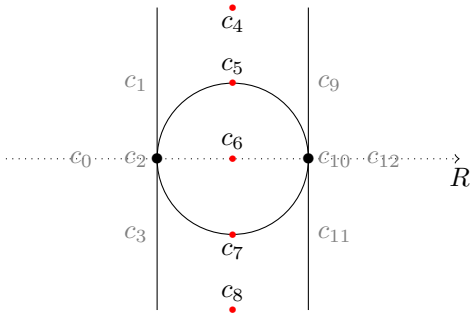
Exemple 5. Exécutons en parallèle ces algorithmes sur l'exemple 1, i.e. en dimension $n = 2$ avec $\mathcal{P} = \{Y^2 + X^2 - 1\}$. Nous admettons que $\text{elim}(\{X^2 + Y^2 - 1\}) = \{4(X^2 - 1)\}$ et $\text{elim}(\{4(X^2 - 1)\}) = \varnothing$. La décomposition en dimension 0 est $\{0\}$ et nous prenons dans cette cellule le point 0 (où 0 désigne le tuple vide). Notons que c'est ici que la zérologie est utile pour unifier ce que nous faisons en dimension 1 et en dimension supérieure. S'il peut paraître étrange de considérer le cylindre au dessus de l'unique point de R^0, cela simplifie beaucoup notre propos. En dimension 1, nous découpons le cylindre $\{0\} \times R = R$ le long des racines de $4(X^2 - 1)$, qui sont -1 et 1. Nous obtenons 5 cellules en dimension 1, et l'Algorithme 2 prend l'échantillon $\{-2, -1, 0, 1, 2\}$ (en rouge ci-dessous).

Input : $n \in \mathbb{N}$ et une partie finie $\mathcal{P}$ de $R[X_0, ..., X_{n-1}]$
Output : Un échantillon S d'une decomposition cylindrique adaptée à $\mathcal{P}$

```
1  if n = 0 then
2  |  S = R^0
3  else
4  |  Soit T = échantillon(elim(P))
5  |  S = ∅
6  |  for x ∈ T do
7  |  |  Soit P_* = {P ∈ P, P⃗(x) ≠ 0}
8  |  |  Soit Q = ∏_{P∈P_*} P⃗(x)
9  |  |  Soit r la suite des racines réelles de Q
10 |  |  if r est vide then
11 |  |  |  S = S ∪ {(x, 0)}
12 |  |  else
13 |  |  |  Soit d = |Q^{-1}({0})| la longueur de r
14 |  |  |  Soit S' obtenue à partir de {r_i, i ∈ [[0, d − 1]]} en ajoutant un élément plus
   |  |  |    petit que tous les éléments de S', un élément plus grand que tous les
   |  |  |    éléments de S', et un élément entre toutes les paires de racines
   |  |  |    consécutives de Q. S = S ∪ S'
15 |  |  end
16 |  end
17 end
18 return S
```

Algorithme 2 : Algorithme échantillon

Posons $\vec{P} : x \mapsto Y^2 + (x^2 - 1)$ l'évaluation partielle de $P = Y^2 + X^2 - 1$ et traitons le découpage de la cellule $]-1, 1[$ de dimension 1. Pour x dans $]-1, 1[$, $\vec{P}(x)$ a deux racines, qui sont $\pm\sqrt{1 - x^2}$. Nous produisons les 5 cellules c_4, c_5, c_6, c_7 et c_8 en dimension 2. Maintenant, pour y choisir des points nous considérons le point qu'on a choisi de la cellule $]-1, 1[$, qui est 0. Pour $x = 0$, $\vec{P}(x) = Y^2 - 1$ a pour racines -1 et 1, donc l'Algorithme 2 choisit les points $(0, -2)$, $(0, -1)$, $(0, 0)$, $(0, 1)$ et $(0, 2)$ (en rouge ci-dessous).



Le simple énoncé de l'Algorithme 1 requiert des justifications non-triviales. En effet, à la ligne 11, il faut s'assurer que le nombre de racines de $Q(y)$ vaut d pour tout $y \in C$ pour décrire ces racines à l'aide de d fonctions. Ces justifications seront données avec la preuve de correction. Dans l'Algorithme 2, la formalisation construit S' en utilisant des opérations algébriques simples sur les éléments de r, e.g. en prenant la moyenne de deux racines consécutives. C'est très peu efficace dans le cas des nombres algébriques. Nous laissons à de futurs travaux le soin de prouver une implémentation rapide de cette opération (e.g. Algorithme 10.109 dans [BPR06]). Les lecteurs les plus observateurs auront remarqué

que, à la ligne 11 de l'Algorithme 1, f_i envoie $y \in R^{n-1} \setminus C$ sur i et non pas sur 0 comme nous pourrions nous y attendre. Nous ne rentrerons pas suffisamment dans les détails pour expliquer pourquoi nous en avons besoin, mais nous devons prouver que $f_0 < \ldots < f_{d-1}$. Les f_i doivent être définies sur R tout entier car C n'est pas un bon modèle de sous-ensemble de R^{n-1}.

3 Correction

La correction de l'Algorithme 2 est obtenue en montrant que celui-ci renvoie au moins un point de chaque cellule du résultat de l'Algorithme 1. Nous ne prouverons pas que l'Algorithme 2 est optimal au sens où il renvoie exactement un point par cellule, bien que ce soit le cas. L'initialisation est triviale pour les deux algorithmes, donc nous nous concentrons sur le cas récursif. Soit donc $n \in \mathbb{N}$ tel que les Algorithmes 1 et 2 soient corrects en dimension n. Soit $\mathcal{P}$ une famille de polynômes de $R[X_0, ..., X_n]$ et soit T le résultat de l'Algorithme 1 sur elim($\mathcal{P}$). T est une CAD adaptée à elim($\mathcal{P}$) d'après l'hypothèse de récurrence, ce qui a un certain nombre de conséquences que nous énonçons ici sans preuve. Soit C une cellule de T.

Proposition 1. *Soient $P, Q \in \mathcal{P}$.*

1. *(`pick`) $C \neq \varnothing$, i.e. nous disposons de pick$(C) \in C$.*
2. *(`nth_const`) Pour tout $i \in \mathbb{N}$, le i-ième coefficient de $\vec{P}(x)$ a un signe constant pour $x \in C$.*
3. *(`S'size`) $x \mapsto deg'(\vec{P}(x))$ est constante sur C.*
4. *(`S'constR`) Le nombre de racines réelles distinctes de $\vec{P}(x)$ est constant pour $x \in C$.*
5. *(`size_gcdpq_const`) $x \mapsto deg'(pgcd(\vec{P}(x), \vec{Q}(x)))$ est constante sur C.*
6. *(`size_gcd_const`) $x \mapsto deg'(pgcd(\vec{P}(x), \vec{P}(x)'))$ est constante sur C.*
7. *Le nombre de racines complexes distinctes de $\vec{P}(x)$ est constant pour $x \in C$.*

Partant de là, nous avons besoin de tels résultats pour le produit des polynômes non nuls de la forme $\vec{P}(\text{pick}(C))$ pour un certain $P \in \mathcal{P}$. C'est la partie centrale de l'argument. Soient $\mathcal{P}_* = \{P \in \mathcal{P}, \vec{P}(\text{pick}(C)) \neq 0\}$, $\vec{\mathcal{P}} = \{\vec{P}, P \in \mathcal{P}_*\}$ et $\tilde{P} = \prod_{P \in \mathcal{P}_*} \vec{P}$. Pour commencer, nous allons montrer que, pour $x, y \in C$ assez proches, $\tilde{P}(x)$ et $\tilde{P}(y)$ ont le même nombre de racines distinctes réelles et complexes en montrant qu'il y a des bijections entre leurs ensembles de racines réelles et complexes. Cela repose sur le théorème de continuité des racines d'un polynôme [NR24], qui dans notre cas s'énonce comme suit.

Lemme 1 (`aligned_deformed`)**.** *Soient $P : R^n[X]$ tel que $\vec{P}$ est continue de degré constant, $x \in R^{n-1}$ et $\delta > 0$. Alors il existe $\varepsilon > 0$ tel que, pour tout $y \in B(x, \varepsilon)$ et pour toute racine u de $\vec{P}(x)$ de multiplicité $m \in \mathbb{N}$, la somme des multiplicités des racines de $\vec{P}(y)$ dans $B(u, \delta)$ soit au moins m.*

La version de ce résultat présente dans [NR24], qui est légèrement plus générale, a été formalisée dans le cadre des travaux présentés ici sans adaptation. On se heurte ici à deux écueils dus aux bibliothèques utilisées. La norme d'un nombre complexe est définie comme étant elle-même un nombre complexe, et non pas un nombre réel comme l'usage mathématique le veut. Cela permet d'utiliser une seule interface pour définir la valeur absolue, le module complexe et la norme, mais cela pose des problèmes car l'ordre sur $R[i]$ ne vérifie aucune bonne propriété, donc il faut constamment revenir à R dès qu'on veut comparer deux normes, ou prendre le minimum d'une famille de réels, par exemple dans les

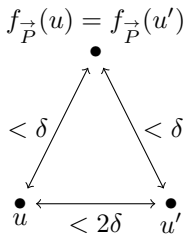
preuves ε-δ de propriétés locales. Un autre problème que l'on rencontre est la difficulté de prouver qu'une fonction est continue. Il faut donner beaucoup d'annotations de type pour que Coq/Rocq trouve les bonnes structures sur lesquelles appliquer les lemmes et un certain nombre de résultats manquent à la librairie.

Grâce au résultat précédent, nous pouvons maintenant établir l'existence d'une bijection entre les racines de $\tilde{P}(x)$ et celles de $\tilde{P}(y)$ pour $x, y \in C$ assez proches. La référence utilisée [BPR06] montre directement que le nombre de racines réelles et complexes de $\tilde{P}(x)$ ne dépend pas de $x \in C$ et ne traite que le cas où $\mathcal{P}$ contient 2 éléments sans expliquer comment traiter le cas général. Le piège est de tenter une récurrence, ce qui ne fonctionne pas (à notre connaissance). Cependant, la preuve se généralise très bien directement. Par ailleurs, les preuves données dans [BPR06] ne mentionnent jamais les bijections sous-jacentes entre l'ensemble des racines de $\tilde{P}(x)$ et de $\tilde{P}(y)$ pour $x, y \in C$ suffisamment proche. Nous adaptons donc les preuves ici en donnant les détails manquant et en traitant directement le cas général. Comme [BPR06], la formalisation prouve directement que le nombre de racines complexes et réelles de $\tilde{P}(x)$ ne dépend pas de $x \in C$. Pour les besoins de l'exposition, on en extrait ici la preuve d'existence de la bijection qui nous intéresse.

Lemme 2. *Soient $x \in C$ et $\delta > 0$. Il existe $\varepsilon > 0$ tel que, pour tout $y \in C \cap B(x, \varepsilon)$, il existe une bijection f entre l'ensemble des racines complexes de $\tilde{P}(x)$ et l'ensemble des racines complexes de $\tilde{P}(y)$ qui induit une bijection entre l'ensemble des racines réelles de $\tilde{P}(x)$ et l'ensemble des racines réelles de $\tilde{P}(y)$ et telle que pour toute racine complexe u de $\tilde{P}(x)$, nous avons $|u - f(u)| < \delta$.*

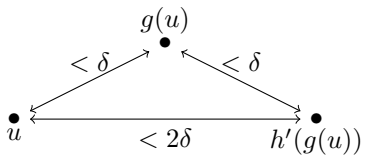
Démonstration Sans perte de généralité, nous pouvons supposer que δ est plus petit que la moitié de la distance entre deux racines complexes distinctes de $\tilde{P}(x)$, de sorte que deux racines complexes distinctes de $\tilde{P}(x)$ à distance au plus 2δ soient égales. Le Lemme 1 appliqué à chaque $\overrightarrow{P} \in \overrightarrow{\mathcal{P}}$ fournit un ensemble de réels strictement positifs dont nous notons ε le minimum. Soit $y \in B(x, \varepsilon)$.

Soit $\overrightarrow{P} \in \overrightarrow{\mathcal{P}}$. Pour toute racine u de $\overrightarrow{P}(x)$, la définition de ε venant du Lemme 1 fournit une racine $f_{\overrightarrow{P}}(u)$ de $\overrightarrow{P}(y)$ telle que $|u - f_{\overrightarrow{P}}(u)| < \delta$. Comme les racines de $\overrightarrow{P}(x)$ (qui sont racines de $\tilde{P}(x)$) sont distantes d'au moins 2δ, $f_{\overrightarrow{P}}$ est injective. Donc, vu que le nombre de racines distinctes de $\overrightarrow{P}(z)$ ne dépend pas de $z \in C$, $f_{\overrightarrow{P}}$ est bijective.



En notant que, pour $z \in C$, l'ensemble des racines de $\tilde{P}(z)$ est l'union des racines des $\overrightarrow{P}(z)$, $\overrightarrow{P} \in \overrightarrow{\mathcal{P}}$, nous obtenons une fonction g de l'ensemble des racines de $\tilde{P}(x)$ vers l'ensemble des racines de $\tilde{P}(y)$ telle que, pour toute racine u de $\tilde{P}(x)$, nous avons $g(u) = f_{\overrightarrow{P}}(u)$ pour un certain $\overrightarrow{P} \in \overrightarrow{\mathcal{P}}$. Nous obtenons de même une fonction h de l'ensemble des racines de $\tilde{P}(y)$ vers l'ensemble des racines de $\tilde{P}(x)$ telle que pour toute racine v de $\tilde{P}(y)$, il existe $\overrightarrow{P} \in \overrightarrow{\mathcal{P}}$ tel que $h'(v) = f^{-1}_{\overrightarrow{P}}(v)$. Nous allons montrer que g et h' sont inverses l'une de l'autre.

Notons déjà que, pour toute racine u de $\tilde{P}(x)$, nous avons $|u - g(u)| < \delta$ et, pour toute racine v de $\tilde{P}(y)$, nous avons $|v - h'(v)| < \delta$. Comme les racines de $\tilde{P}(x)$ sont distantes d'au moins 2δ, $h' \circ g$ est l'identité.



Nous remarquons que, pour $\overrightarrow{P} \in \overrightarrow{\mathcal{P}}$ et v une racine de $\overrightarrow{P}(y)$, v est la seule racine de $\overrightarrow{P}(y)$ à distance au plus δ de $\overrightarrow{P}(y)$. En effet, si v' est une telle racine, alors $h'(v)$ et $f^{-1}_{\overrightarrow{P}}(v')$ sont deux racines de $\tilde{P}(x)$ à distance au plus 2δ, donc sont égaux. Pour $v' = v$, nous obtenons

aussi $h'(v) = f^{-1}_{\vec{P}}(v)$ et l'injectivité de $f^{-1}_{\vec{P}}$ permet de conclure $v' = v$.

Montrons que h' conserve la multiplicité, au sens où, pour tous $\vec{P} \in \vec{\mathcal{P}}$ et v une racine de $\vec{P}(y)$, la multiplicité de $h'(v)$ comme racine de $\vec{P}(x)$ est la même que la multiplicité de v comme racine de $\vec{P}(y)$. Soient donc $\vec{P} \in \vec{\mathcal{P}}$ et v une racine de $\vec{P}(y)$. v est la seule racine de $\vec{P}(y)$ à distance au plus δ de $h'(v)$. D'après le Lemme 1, la multiplicité de v comme racine de $\vec{P}(y)$ est au moins la multiplicité de $h'(v)$ comme racine de $\vec{P}(x)$. Ceci est valable pour toute racine v de $\vec{P}(y)$ et h' est surjective. Or $\vec{P}(x)$ et $\vec{P}(y)$ ont le même degré, et donc le même nombre de racines comptées avec multiplicités, donc l'égalité précédente est une égalité.

Montrons à présent que $g \circ h'$ est l'identité. Soit v une racine de $\tilde{P}(y)$. Soient $\vec{P}, \vec{Q} \in \vec{\mathcal{P}}$ tels que v est une racine de $\vec{P}(y)$ et $g(h'(v))$ est une racine de $\vec{Q}(y)$. Nous avons vu que $h' \circ g$ est l'identité, d'où l'on tire que $h'(g(h'(v)) = h'(v)$ est une racine à la fois de $\vec{P}(x)$ et $\vec{Q}(x)$. Or, h' conserve la multiplicité, donc v est aussi une racine de $\vec{Q}(y)$. Comme il n'y a qu'une seule racine de $\vec{Q}(y)$ à distance au plus δ de $h'(v)$, nous obtenons $g(h'(v)) = v$.

Passons maintenant au cas réel : il reste à montrer que g induit une bijection entre les racines réelles de $\tilde{P}(x)$ et les racines réelles de $\tilde{P}(y)$. Soit u une racine réelle de $\tilde{P}(x)$. Comme $\tilde{P}(y)$ est à coefficients réels, $\overline{g(u)}$ est une racine de $\tilde{P}(y)$, et donc $g^{-1}(\overline{g(u)})$ est une racine de $\tilde{P}(x)$. De $|u - g(u)|$ et du fait que u est réel, nous tirons $|u - \overline{g(u)}| < \delta$, puis $|u - g^{-1}(\overline{g(u)})| < 2\delta$. D'où $g^{-1}(\overline{g(u)}) = u$, ou encore $\overline{g(u)} = g(u)$. Nous obtenons que $g(u)$ est réel. Soit maintenant v une racine de $\tilde{P}(y)$. $g^{-1}(v)$ est une racine de $\tilde{P}(x)$, qui est à coefficients réels, donc $\overline{g^{-1}(v)}$ est aussi une racine de $\tilde{P}(x)$. Comme v est réel et $|v - g^{-1}(v)| < \delta$, nous avons $|v - \overline{g^{-1}(v)}| < \delta$, d'où l'on tire $|g^{-1}(v) - \overline{g^{-1}(v)}| < 2\delta$, puis $\overline{g^{-1}(v)} = g^{-1}(v)$. Ceci montre que $g^{-1}(v)$ est réel. En conclusion, g induit bien une bijection entre les racines réelles de $\tilde{P}(x)$ et les racines réelles de $\tilde{P}(y)$. □

Le fait de devoir donner tous les détails de la preuve a permis de faire apparaître l'argument selon lequel h' conserve la multiplicité, qui factorise une preuve déjà très longue. Une difficulté pratique qui est apparue ici est le fait de manipuler des ensembles d'objets. La bibliothèque qui définit ces objets est très lente. En particulier, la moindre erreur peut lancer Coq/Rocq dans des calculs extrêmement longs, ce qui oblige l'utilisateur à interrompre manuellement l'assistant de preuve et à redémarrer la compilation de tout le fichier de preuve.

On déduit du résultat précédent que le nombre de racines réelles et complexes distinctes de $\tilde{P}(x)$ est localement constant. Cependant, nous n'avons pas supposé que le corps réel clos sur lequel nous travaillons est connexe. Ceci pose problème car la connexité des C donnerait gratuitement le fait que le nombre de racines distinctes de $\tilde{P}(x)$ est constant pour $x \in C$. Il existe une variante de la notion de connexité, appelée connexité semi-algébrique, où l'on demande que le recouvrement soit fait par des parties semi-algébriques. Cette propriété est suffisante pour retrouver que les fonctions localement constantes sont constantes, mais seulement pour les fonctions semi-algébriques. Le fait que le découpage soit fait par des fonctions semi-algébriques dans la définition de cellule implique que C est semi-algébrique. Le résultat dont nous avons besoin devient :

Lemme 3 (`SAconnected_locally_constant_constant` [BPR06])**.** *Soient $m \in \mathbb{N}$ et $f : C \to R^m$ semi-algébrique. Si f est localement constante sur C, alors f est constante sur C.*

Il ne reste plus qu'à prouver que les fonctions donnant le nombre de racines réelles et complexes distinctes d'un polynôme sont semi-algébriques. Cela ne fait a priori pas sens puisque ces fonctions vont de $R[X]$ dans R, mais nous pouvons encoder un polynôme de degré d dans R^{d+1} avec la fonction $P \mapsto (P_0, ..., P_d)$, qui à un polynôme associe la liste de ses $d+1$ premiers coefficients, d'inverse $(x_0, ..., x_d) \mapsto \sum_{i=0}^{d} x_i X^i$. Autrement dit, on représente

un polynôme de degré au plus d par l'élément de R^{d+1} dont les coordonnées sont données par les coefficients du polynôme. Ensuite, la clé du résultat est le schéma de formules suivant, indexé par n et $m \in \mathbb{N}$, qui décrit les racines réelles d'un polynôme [BPR06]. Dans cette formule, $X_0, ..., X_{n-1}$ encode un polynôme de degré au plus n, m est le nombre de racines et $X_n, .., X_{n+m-1}$ les racines auxquelles nous nous attendons, et eval_n est une formule décrivant l'évaluation d'un polynôme en un point, où $X_0, ..., X_{n-1}$ décrit le polynôme, X_n décrit le point d'évaluation et X_{n+1} la valeur de retour. En d'autres termes, eval_n représente le graphe de la fonction d'évaluation.

$$
\begin{aligned}
&(\bigwedge_{i=0}^{n-1} X_i = 0) \wedge m = 0) \\
&\quad \vee \quad (\bigwedge_{i=n}^{n+m-1} \text{eval}_n[X_n \leftarrow X_i, X_{n+1} \leftarrow 0]) \\
&\qquad \wedge \quad (\bigwedge_{i=n}^{n+m-2} X_i < X_{i+1}) \\
&\qquad \wedge \quad (\forall X_{n+m}, \text{eval}_n[X_n \leftarrow X_{n+m}, X_{n+1} \leftarrow 0] \\
&\qquad\qquad \rightarrow \bigvee_{i=n}^{n+m-1} X_{n+m} = X_i).
\end{aligned}
$$

La première condition traite à part le cas du polynôme nul, pour lequel on décide par convention que le nombre de racines est nul. Le cas des polynômes non-nul est traité par les trois conditions 4suivantes. La première affirme que le polynôme s'annule en X_i pour tout i entre n et $n + m - 1$, la deuxième que les racines sont données dans l'ordre strictement croissant et la dernière que le polynôme n'a pas de racine en dehors de $\{X_i, n \leq i \leq n+m-1\}$. La deuxième condition est trop forte, car il suffit que les X_i, pour i entre n et $n + m - 1$ soient distinctes, mais les opérations spécifiées dans *MathComp/real-closed* donnent les racines réelles d'un polynôme dans l'ordre croissant. Cela ne nous coûte rien et simplifie à la fois l'énoncé et son utilisation, car *MathComp* contient de nombreux résultats utiles sur les listes ordonnées que nous pouvons directement appliquer.

Le degré d de $\tilde{P}(x)$ étant constant sur C, nous pouvons l'encoder dans R^{d+1} et nous obtenons :

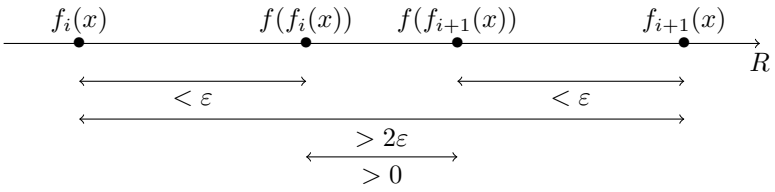
Corollaire 1 (`S'const` et `size_gcdpm_const` [BPR06])**.** *Le nombre de racines réelles et complexes distinctes de $\tilde{P}(x)$ est constant pour $x \in C$.*

Nous avons maintenant les fonctions dont nous avons besoin pour découper le cylindre $C \times R$. Ces fonctions sont semi-algébriques en vertu de la formule décrivant les racines d'un polynôme décrite ci-dessus. Il ne reste qu'à montrer qu'elles sont continues. [BPR06] affirme que c'est le cas comme conséquence immédiate du Lemme 2. Cependant, procéder ainsi complexifie encore la partie la plus difficile de l'argumentation. Nous avons donc choisi de traiter un cas légèrement plus général :

Lemme 4 (`rootsR_continuous`)**.** *Soit $P : C \to R[X]$ continue, de degré d constant et de nombre de racines réelles et complexes distinctes constant. Soient $f_1 \leq ... \leq f_k$ les fonctions décrivant les racines de P. Alors, pour tout $i \in [\![1, k]\!]$, f_i est continue.*

Démonstration Soient $x \in C$ et $\varepsilon > 0$. Sans perte de généralité, nous pouvons supposer que ε est inférieur à la moitié de la distance entre deux racines complexes distinctes de $P(x)$. Le Lemme 1 fournit $\delta > 0$ tel que pour tout $y \in C \cap B(x, \delta)$ et pour toute racine u de $P(x)$, il existe une racine de $P(y)$ dans $B(u, \varepsilon)$. Soit $y \in C \cap B(\delta, x)$. Pour toute racine u de $P(x)$, nous avons une racine $f(u)$ de $P(y)$ telle que $|u - f(u)| < \varepsilon$. Comme deux racines de $P(x)$ sont distantes d'au moins 2ε, f est injective. Or $P(x)$ et $P(y)$ ont le même nombre de racines complexes distinctes, donc f est bijective. Si v est une racine réelle de $P(y)$, alors, comme $P(x)$ est à coefficients réels, $f^{-1}(v)$ et $\overline{f^{-1}(v)}$ sont deux racines de $P(x)$ à distance inférieure à 2δ, donc sont égaux. Ceci montre que f^{-1} induit une fonction

injective de l'ensemble des racines réelles de $P(y)$ dans l'ensemble des racines réelles de $P(x)$. Comme $P(x)$ et $P(y)$ ont le même nombre de racines réelles distinctes, f^{-1} induit même une bijection. Comme les $f_i(x)$, $i \in [\![1, k]\!]$ sont distants d'au moins 2ε, f est croissante : les inégalités $f_1(x) < ... < f_k(x)$ impliquent $f(f_1(x)) < ... < f(f_k(x))$. Alors, pour tout $i \in [\![1, k]\!]$, nous avons $|f_i(y) - f_i(x)| = |f(f_i(x)) - f_i(x)| < \varepsilon$, ce qui conclut la preuve.



□

Nous pouvons faire le découpage de $C \times R$. Pour montrer qu'on obtient une CAD, il reste à montrer que les découpages des cellules de T produisent bien une partition de R^{n+1}. Le texte mathématique [BPR06] ne montre jamais ce point, qui est visuellement évident pour un expert, mais décrire cette intuition dans le cadre d'une preuve formelle n'est pas évident. Notons S l'ensemble des cellules de R^{n+1} obtenues. Montrer qu'elles sont non-vides et qu'elles recouvrent R^n ne pose pas de problème. Montrer que S possède la propriété de l'intersection triviale est plus difficile. En effet, il y a $O(|S|^2)$ contraintes à vérifier. Dans le contexte abstrait où nous sommes, nous traitons ensemble les cellules qui ont été construites de la même manière, mais cela laisse tout de même 16 cas à traiter. Une première astuce consiste à utiliser le fait que T est lui-même une partition. Ainsi, avec $\pi : R^{n+1} \to R^n$ la projection canonique, si $C, D \in S$ vérifient $\pi(C) \neq \pi(D)$, nous avons tout de suite $\pi(C) \cap \pi(D) = \varnothing$, puis $C \cap D = \varnothing$. Une deuxième astuce consiste à ordonner les cellules avec l'ordre suivant. Pour $X, Y \subset R^{n+1}$, nous définissons $X < Y$ par :

$$X < Y \Leftrightarrow Y \neq \varnothing \wedge \forall x \in R^n, \forall y, z \in R, (x, y) \in X \Rightarrow (x, z) \in Y \Rightarrow y < z$$

Autrement dit, pour $X, Y \subset R^{n+1}$, on a $X < Y$ lorsque tout point $y \in Y$ est strictement au dessus de tous les points de X avec les mêmes premières coordonnées que y.

Montrer que le découpage de $C \times R$ produit des cellules ordonnées revient à montrer que deux cellules consécutives le sont, ce qui réduit le nombre de contraintes à $O(|S|)$. Plus précisément, il y 4 cas à traiter. En conclusion, nous obtenons le Lemme suivant.

Lemme 5 (`elimp_lift_CD` [BPR06])**.** *S est une CAD de R^{n+1}.*

Tout ce qui précède montre que l'Algorithme 1 est bien défini. Pour montrer la correction, il suffit de montrer que la partition produite est bien adaptée à $\mathcal{P}$. Ce point est encore considéré trivial par un mathématicien [BPR06], car, avec C une cellule de S, si $f_1 < ... < f_k : C \to R$ décrivent les racines de $\tilde{P}$, alors les $f_i, 1 \leq i \leq k$ sont de multiplicité constantes sur C et le signe de $\tilde{P}$ en un point (x, y) ne dépend que des multiplicités des racines de $\tilde{P}(x)$ dans $]-\infty, y[$. Cependant, cet argument est pénible à formaliser car il requiert encore de construire des bijections entre les ensembles de racines de certains polynômes. Nous lui avons donc préféré une utilisation du théorème des valeurs intermédiaires, qui est encore valable pour des ensemble semi-algébriquement connexes.

Proposition 2 (`elimp_lift_adapted` [BPR06])**.** *S est adaptée à $\mathcal{P}$.*

Démonstration Soient $P \in \mathcal{P}$, $C \in S$ et $D = \pi(C)$. Nous souhaitons montrer que P est de signe constant sur C. Si $\overrightarrow{P}(\text{pick}(D)) = 0$, alors $\overrightarrow{P}$ envoie tous les points de D sur 0 et le résultat est clair. Supposons donc que $\overrightarrow{P}(\text{pick}(D)) \neq 0$. Nous distinguons deux cas, selon que P s'annule ou non sur C.

— Supposons que, pour tout $x \in C$, nous avons $P(x) \neq 0$. Nous savons montrer que $P(C)$ est semi-algébriquement connexe grâce au fait que C l'est et que P induit une fonction semi-algébrique continue de C dans R. Comme $P(C)$ ne contient pas 0, il est inclus dans $]-\infty, 0[$ ou $]0, +\infty[$, ce qui conclut la preuve.
— Supposons qu'on dispose de $x \in C$ tel que $P(x) = 0$. Alors x_n est une racine de $\vec{P}(\pi(x))$, donc de $\tilde{P}(\pi(x))$, disons la i-ième. Alors, avec $f_1 < ... < f_k : D \to R$ décrivant les racines de $\tilde{P}$ sur D, le graphe de f_i est une cellule de S contenant x, et est donc C. Soient $g_1 < ... < g_l : D \to R$ décrivant les racines de $\vec{P}$. Nous savons montrer que $(g_j)_{j \in [\![1,l]\!]}$ est une sous-suite de $(f_j)_{j \in [\![1,l]\!]}$, de sorte qu'il existe $j \in [\![1, l]\!]$ tel que $f_i = g_j$. P s'annule sur tout le graphe de g_j, c'est-à-dire sur tout le graphe de f_i, ou encore sur C, ce qui conclut la preuve.

□

Nous obtenons la correction de l'Algorithme 1. En ce qui concerne l'Algorithme 2, si nous notons T' et S' ses résultats sur elim($\mathcal{P}$) et $\mathcal{P}$, sa correction en dimension n implique que T' contient au moins un point de chaque cellule de T. Vu la façon dont nous avons construit S et dont nous choisissons les points de S', S' contient au moins un point par cellule de S.

Théorème 1 (`cylindrical_decomposition` et `sample_cylindrical_decompositionP`)**.** *Les Algorithmes 1 et 2 sont corrects.*

Dans l'assistant de preuve, ce résultat s'écrit comme suit. L'Algorithme 1 est inclus dans l'énoncé de son théorème de correction, au sens où ce dernier affirme constructivement l'existence de la CAD. Le deuxième énoncé est la spécification de l'Algorithme 2, nommé `sample_cylindrical_decomposition` dans la formalisation.

```
Theorem cylindrical_decomposition n (P : {fset {mpoly R[n]}}) :
  { S | isCylindricalDecomposition S /\
        forall p : P, poly_adapted (val p) S}.

Lemma sample_cylindrical_decompositionP n (P : {fset {mpoly R[n]}}) :
  exists S, isCylindricalDecomposition S /\
    (forall p : P, poly_adapted (val p) S) /\
    forall s : S, exists x : sample_cylindrical_decomposition P,
      (val x) \in (val s).
```

4 Conclusion

Cet article décrit la première (au mieux de nos connaissances) formalisation d'un algorithme de calcul de Décomposition Algébrique Cylindrique. Il repose sur la bibliothèque *MathComp* pour les définitions et théories des objets mathématiques sous-jacents. La preuve de correction est supposément constructive, mais nous utilisons la bibliothèque *MathComp-Analysis* pour la notion de continuité et cette bibliothèque impose l'utilisation d'axiome classiques. Cependant, nous n'utilisons essentiellement que la définition de continuité, donc nous pensons que ces axiomes ne sont pas réellement nécessaires, au sens où nous pourrions réécrire la partie de la bibliothèque dont nous avons besoin sans ces axiomes. Pour obtenir une implémentation vérifiée de l'algorithme, il suffit de fournir une implémentation efficace de certains composants, comme le calcul des sous-résultants de deux matrices ou l'encodage de Thom pour les calculs sur les nombres algébriques, et de prouver que ces composants sont équivalents à ce qui existe dans *MathComp*.

Remerciements. Merci à Yves Bertot et Cyril Cohen pour leur soutien et les nombreuses discussions que nous avons eues.

Références

[BOKR86] Michael Ben-Or, Dexter Kozen et John Reif : The complexity of elementary algebra and geometry. *Journal of Computer and System Sciences*, 32(2):251–264, 1986.

[BPR06] Saugata Basu, Richard Pollack et Marie-Françoise Roy : *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[Col75] George E. Collins : Quantifier elimination for real closed fields by cylindrical algebraic decompostion. *In* H. Brakhage, éditeur : *Automata Theory and Formal Languages*, pages 134–183, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg.

[Dja18] Boris Djalal : A Constructive Formalisation of Semi-algebraic Sets and Functions. *In* June Andronick et Amy Felty, éditeurs : *CPP 2018 - Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 240–251, Los Angeles, California, United States, janvier 2018. ACM.

[HJX16] Jingjun Han, Zhi Jin et Bican Xia : Proving inequalities and solving global optimization problems via simplified cad projection. *Journal of Symbolic Computation*, 72:206–230, 2016.

[KTP23] Katherine Kosaian, Yong Kiam Tan et André Platzer : A first complete algorithm for real quantifier elimination in isabelle/hol. *In Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2023, page 211–224, New York, NY, USA, 2023. Association for Computing Machinery.

[Mah05] Assia Mahboubi : Programming and certifying a CAD algorithm in the coq system. *In* Thierry Coquand, Henri Lombardi et Marie-Françoise Roy, éditeurs : *Mathematics, Algorithms, Proofs, 9.-14. January 2005*, volume 05021 de *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

[MC12] Assia Mahboubi et Cyril Cohen : Formal proofs in real algebraic geometry : from ordered fields to quantifier elimination. *Logical Methods in Computer Science*, Volume 8, Issue 1, février 2012.

[MND18] César A. Muñoz, Anthony J. Narkawicz et Aaron Dutle : A decision procedure for univariate polynomial systems based on root counting and interval subdivision. *J. Formaliz. Reason.*, 11(1):19–41, 2018.

[NR24] Melvyn B. Nathanson et David A. Ross : Continuity of the roots of a polynomial. *Communications in Algebra*, 52(6):2509–2518, 2024.

[Pla22] André Platzer : Correction to : Differential dynamic logic for hybrid systems. *J. Autom. Reason.*, 66(1):173, 2022.

[SS83] Jacob T Schwartz et Micha Sharir : On the "piano movers" problem. ii. general techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4(3):298–351, 1983.

[ST11] Thomas Sturm et Ashish Tiwari : Verification and synthesis using real quantifier elimination. *In Proceedings of the 36th International Symposium on Symbolic and Algebraic Computation*, ISSAC '11, page 329–336, New York, NY, USA, 2011. Association for Computing Machinery.