Initiation à la recherche

M1 - Mathématique fondamentale et appliquée

Victor Locherer - 21913014

Université Paris Cité - Année 2024/2025

Ce court document présente les travaux effectués dans le cadre de mon initiation à la recherche de M1 Mathématiques fondamentales de l'Université Paris-Cité.

L'enseignant-chercheur encadrant cette initiation est Ricardo Brasca.

Le suivi s'est effectué en présentiel et à distance, par mail et avec des rendez-vous de suivi hebdomadaire.

Le sujet de cette initiation est l'assistant de preuve lean et de la bibliothèque mathématique associée. Pour cette introduction à la formalisation assistée par ordinateur, un théorème du cours d'Arithmétique (par Marc Hindry) suivi cette année à été choisi.

La théorème s'énonce comme suit :

Soit F_q est un corps fini de caractéristique p, et n un entier premier avec p. si r est l'ordre de q modulo n, alors le n-ième polynôme cyclotomique se décompose en $\frac{\Phi(n)}{r}$ polynômes de degré r.

La preuve utilisée :

La preuve initiale du cours de Marc Hindry était peu adaptée pour une formalisation dans lean, nous sommes donc partis sur une autre preuve, plus Galoisienne :

Soit k le degré d'un polynôme irréductible P divisant le n-ième polynôme cyclotomique dans F_q. Alors k est aussi le degré de l'extension galoisienne de F_q vers L, le corps de décomposition de P. Autrement dit, c'est le cardinal du groupe de Galois associé à cette extension. Or, la théorie des corps finis nous donne que le Frobenius $\phi : x \to x^q$ est un générateur du groupe de galois. En explicitant cette relation, avec la base formée par les racines de P, on obtient que l'ordre de $q \bmod n$ (n étant par définition l'ordre des dites racines) est égal à k. Pour pouvoir bien expliciter les détails de la démonstration à l'assistant de preuve, une égalité par double division a été utilisée, en profitant toujours du fait que l'ordre du frobenius dans $Gal(L/F_q)$ est k.

Cela nous donne donc que k divise l'ordre de $q \bmod n$ (puisque q puissance k vaut 1 mod n), et que l'ordre de $q \bmod n$ divise k (puisque k est l'ordre du Frobenius). Le polynôme P étant choisi arbitrairement, il vient alors trivialement que le n-ième polynôme cyclotomique se décompose en $\phi(n)/r$ de ces polynômes.

La transcription en Lean :

La première partie du code lean est entièrement dédiée à introduire les objets utilisés, et à justifier les nombreux détails que requiert une preuve formalisée en assistant de preuve.

Lean étant un langage construit sur la théorie des types, de nombreux choix pratiques ont dû être fait lors de la construction de la bibliothèque mathlib, afin que celle-ci puisse être suffisamment abstraite et générique pour de nombreux usages. Ces complications inévitables propres à la théorie des types amène à des absurdités et des contractions désagréables lors ce qu'il s'agit de montrer des choses considérées comme triviales dans une démonstration classique sur papier.

Par exemple, afin de montrer qu'une racine de P est une racine primitive, (hypothèse $h\zeta'$, ligne 67), le théorème isRoot_cyclotomic_iff de la bibliothèque mathlib nous permet de conclure en quatre lignes le fait relativement évident que les racines d'un polynôme divisant un polynôme cyclotomique sont des racines primitives de l'unité.

```
have hζ' : IsPrimitiveRoot ζ' n := by
   rw [← isRoot_cyclotomic_iff]
   exact IsRoot.dvd
     (AdjoinRoot.isRoot_root Q)
     (by simpa using Polynomial.map_dvd (algebraMap K L) hQ)
```

Toutefois, ce théorème requiert d'avoir comme hypothèse que n ne soit pas zéro dans L. Cela nous paraît évident mathématiquement puisque que l'hypothèse que q soit premier avec n est là pour que l'énoncé ait du sens, et amènerait un autre énoncé tout autre si ce n'était pas le cas. Mais Lean à besoin de cette hypothèse démontrée pour valider la preuve. Ainsi, la démonstration que n n'est pas zéro dans L, elle, prend 20 lignes pour être réalisée, avec notamment le traitement du cas absurde où $q = 1$.

```
have n0 : NeZero (n : L) := by
     suffices NeZero (n : K) by
       simpa using NeZero.of_injective (algebraMap K (AdjoinRoot Q)).injective
     have hf : 0 < f := by
       apply Nat.pos_of_ne_zero
       intro hf
       simp [hf] at hK
       rw [Fintype.card_eq_one_iff] at hK
       obtain ⟨x, hx⟩ := hK
       have : (0 : K) ≠ 1 := by
         exact zero_ne_one' K
       have : (0 : K) = 1 := by
         rw [hx 0, hx 1]
       contradiction
     have : CharP K p := by
       rw [CharP.charP_iff_prime_eq_zero hp.1]
       have hf1 : f ≠ 0 := by
         exact Nat.ne_zero_of_lt hf
       simpa [hf1, hK] using Nat.cast_card_eq_zero K
     apply NeZero.of_not_dvd K (p := p)
     rw [← Nat.Prime.coprime_iff_not_dvd hp.1]
     exact (Nat.coprime_pow_left_iff hf _ _).mp hn
```

La démonstration commence alors à la ligne 90, avec le premier rewrite. Cela correspond à l'énoncé "k est le degré de l'extension L/F_q, donc le cardinal du groupe de Galois associé, autrement dit l'ordre du frobenius".

```
 rw [← AdjoinRoot.powerBasis_dim this.elim.ne_zero, ← pB.finrank,←
FiniteField.orderOf_frobeniusAlgEquivOfAlgebraic]
```

Vient alors l'application du lemme dvd_antisymm (ligne 92), qui sépare l'objectif avec égalité en deux objectifs avec division.

```
 apply dvd_antisymm
```

Les deux objectifs prennent alors moins de 20 lignes chacun à être réalisés, mathlib et les lemmes de la première partie de la démonstration nous donnant l'essentiel du matériel nécessaire.

Démonstration que l'ordre de q modulo n divise k :

```
 ---P1
 rw [orderOf_dvd_iff_pow_eq_one]
 set φ := (FiniteField.frobeniusAlgEquivOfAlgebraic K L)
 have : (φ ^ orderOf φ) ζ' = ζ' := by simp [pow_orderOf_eq_one φ]
 simp only [AlgEquiv.coe_pow, φ,
FiniteField.coe_frobeniusAlgEquivOfAlgebraic, pow_iterate, hK] at this
 ext
 set o := orderOf (FiniteField.frobeniusAlgEquivOfAlgebraic K (AdjoinRoot
Q))
 simp
 norm_cast
 rw [← Nat.cast_one, ZMod.eq_iff_modEq_nat, IsPrimitiveRoot.eq_orderOf hζ]
 have : ζ ^ (p^f) ^ o = ζ := by
   ext
   simpa
 nth_rewrite 2 [← pow_one ζ] at this
 exact pow_eq_pow_iff_modEq.mp this
```

Démonstration que k divise l'ordre de q modulo n :

```
 ---P2
 rw [orderOf_dvd_iff_pow_eq_one]
 apply AlgEquiv.coe_algHom_injective
 apply pB.algHom_ext
 simp [FiniteField.coe_frobeniusAlgEquivOfAlgebraic, hK, pB]
 set φ := (FiniteField.frobeniusAlgEquivOfAlgebraic K L)
 have : (φ ^ orderOf φ) ζ' = ζ' := by simp [pow_orderOf_eq_one φ]
 simp only [AlgEquiv.coe_pow, φ, FiniteField.coe_frobeniusAlgEquivOfAlgebraic,
pow_iterate, hK] at this
 suffices ζ ^ (p^f) ^ orderOf (unitOfCoprime (p^f) hn) = ζ by
   exact Units.eq_iff.2 this
 nth_rewrite 2 [← pow_one ζ]
 rw [pow_eq_pow_iff_modEq, ← IsPrimitiveRoot.eq_orderOf hζ, ←
ZMod.eq_iff_modEq_nat]
 have : (unitOfCoprime (p^f) hn) ^ (orderOf ((unitOfCoprime (p^f) hn))) = 1 :=
by
   exact pow_orderOf_eq_one (unitOfCoprime (p^f) hn)
 simp [← Units.eq_iff] at this
 push_cast
 exact this
```

L'assistant de preuve ne donnant alors plus d'erreurs, et le programme ne contenant pas de sorry, le théorème est alors prouvé par vérification par ordinateur. On peut alors être certain de la véracité du théorème dans le set d'axiome utilisé par lean. C'est l'un des avantages des assistants de preuve, la possibilité d'une erreur une fois la preuve validée par ordinateur est restreinte à une mauvaise formulation de l'objectif (et auquel cas, on a bien démontré quelque chose, mais ce quelque chose n'était pas ce que l'on pensait être, ce qui n'est pas tellement un bug ou erreur de code tel que l'on pourrait les avoir dans de la programmation plus classique).

Voici sur les trois prochaines pages le code entier :

```
import Mathlib

variable {K : Type*} [Field K] [Fintype K]
variable {p f n : ℕ} [hp : Fact p.Prime]

(hK : Fintype.card K = p^f)
(hn : (p^f).Coprime n)

open Polynomial ZMod
open Prime

include hK

theorem foo {P : K[X]} (hP : P ∣ cyclotomic n K) (hPirr : Irreducible P) :
   orderOf (unitOfCoprime _ hn) = P.natDegree := by

 classical
 let Q := normalize P

 let hQmo : Monic Q := by
   apply P.monic_normalize hPirr.ne_zero

 let L := AdjoinRoot Q

 have : Module.Finite K L := hQmo.finite_adjoinRoot
 have : Finite L := Module.finite_of_finite K

 have : Fact (Irreducible Q) := by
   have qIrr : Irreducible Q := by
     suffices Irreducible P by
       apply Associated.irreducible (p:=P)
       apply associated_normalize P
       exact this
     exact hPirr
   exact ⟨qIrr⟩

 have hQ : Q ∣ cyclotomic n K := by
   apply dvd_trans (b:=P)
   rw[normalize_dvd_iff]
   exact hP

 let ζ' := AdjoinRoot.root Q
```

```
have n0 : NeZero (n : L) := by
    suffices NeZero (n : K) by
      simpa using NeZero.of_injective (algebraMap K (AdjoinRoot Q)).injective
    have hf : 0 < f := by
      apply Nat.pos_of_ne_zero
      intro hf
      simp [hf] at hK
      rw [Fintype.card_eq_one_iff] at hK
      obtain ⟨x, hx⟩ := hK
      have : (0 : K) ≠ 1 := by
        exact zero_ne_one' K
      have : (0 : K) = 1 := by
        rw [hx 0, hx 1]
      contradiction
    have : CharP K p := by
      rw [CharP.charP_iff_prime_eq_zero hp.1]
      have hf1 : f ≠ 0 := by
        exact Nat.ne_zero_of_lt hf
      simpa [hf1, hK] using Nat.cast_card_eq_zero K
    apply NeZero.of_not_dvd K (p := p)
    rw [← Nat.Prime.coprime_iff_not_dvd hp.1]
    exact (Nat.coprime_pow_left_iff hf _ _).mp hn

have hζ' : IsPrimitiveRoot ζ' n := by
  rw [← isRoot_cyclotomic_iff]
  exact IsRoot.dvd
    (AdjoinRoot.isRoot_root Q)
    (by simpa using Polynomial.map_dvd (algebraMap K L) hQ)

have hζ0 : ζ' ≠ 0 := by
  apply IsPrimitiveRoot.ne_zero (hζ')
  exact (NeZero.of_neZero_natCast (h := n0)).ne

let ζ := Units.mk0 ζ' hζ0

have hζ : IsPrimitiveRoot ζ n := by
  exact IsPrimitiveRoot.coe_units_iff.mp hζ'

let pB := AdjoinRoot.powerBasis this.elim.ne_zero

have hDeg : P.natDegree = Q.natDegree := by
  apply P.natDegree_eq_of_degree_eq
  rw[P.degree_normalize]
```

```
 rw [hDeg]

 rw [← AdjoinRoot.powerBasis_dim this.elim.ne_zero, ← pB.finrank,←
FiniteField.orderOf_frobeniusAlgEquivOfAlgebraic]

 apply dvd_antisymm

 ---P1
 rw [orderOf_dvd_iff_pow_eq_one]
 set φ := (FiniteField.frobeniusAlgEquivOfAlgebraic K L)
 have : (φ ^ orderOf φ) ζ' = ζ' := by simp [pow_orderOf_eq_one φ]
 simp only [AlgEquiv.coe_pow, φ, FiniteField.coe_frobeniusAlgEquivOfAlgebraic,
pow_iterate, hK] at this
 ext
 set o := orderOf (FiniteField.frobeniusAlgEquivOfAlgebraic K (AdjoinRoot Q))
 simp
 norm_cast
 rw [← Nat.cast_one, ZMod.eq_iff_modEq_nat, IsPrimitiveRoot.eq_orderOf hζ]
 have : ζ ^ (p^f) ^ o = ζ := by
   ext
   simpa
 nth_rewrite 2 [← pow_one ζ] at this
 exact pow_eq_pow_iff_modEq.mp this

 ---P2
 rw [orderOf_dvd_iff_pow_eq_one]
 apply AlgEquiv.coe_algHom_injective
 apply pB.algHom_ext
 simp [FiniteField.coe_frobeniusAlgEquivOfAlgebraic, hK, pB]
 set φ := (FiniteField.frobeniusAlgEquivOfAlgebraic K L)
 have : (φ ^ orderOf φ) ζ' = ζ' := by simp [pow_orderOf_eq_one φ]
 simp only [AlgEquiv.coe_pow, φ, FiniteField.coe_frobeniusAlgEquivOfAlgebraic,
pow_iterate, hK] at this
 suffices ζ ^ (p^f) ^ orderOf (unitOfCoprime (p^f) hn) = ζ by
   exact Units.eq_iff.2 this
 nth_rewrite 2 [← pow_one ζ]
 rw [pow_eq_pow_iff_modEq, ← IsPrimitiveRoot.eq_orderOf hζ, ←
ZMod.eq_iff_modEq_nat]
 have : (unitOfCoprime (p^f) hn) ^ (orderOf ((unitOfCoprime (p^f) hn))) = 1 :=
by
   exact pow_orderOf_eq_one (unitOfCoprime (p^f) hn)
 simp [← Units.eq_iff] at this
 push_cast
 exact this
```

En conclusion :

Lean est un outil formidable, exigeant certes, mais ne laissant pas de place possible au doute ou à l'erreur dans la compréhension des théorèmes prouvés. Beaucoup de choses excessivement triviales peuvent encore prendre beaucoup de temps à être démontrés en Lean aujourd'hui pour quelqu'un en découverte du logiciel. Mais la bibliothèque mathlib et sa communauté sur Zulip grandissent, et la richesse actuelle de l'outil permet, une fois ce cap d'introduction terminé, de prendre facilement du plaisir durant ces formalisations.

Cela a aussi été pour moi une bonne façon de pratiquer des théorèmes vus pendant l'année, et de nombreux projets dans mes sujets de prédilections sont actuellement actifs sur Lean (théorie algébrique des nombres, le théorème de Fermat, que ce soit pour des petites valeurs ou pour le cas général, représente une part active de la communauté autour de lean et mathlib).

Ce théorème montré ici à lui même un lien avec Fermat, permettant à Riccardo Brasca de montrer que l'anneau des entiers liés du 7ème corps cyclotomique est principal, ce qui permet par la théorie de Kummer de conclure le théorème de Fermat dans le cas n=7.

Pour conclure, j'aimerai remercier Riccardo Brasca pour son temps et son aide, me fournissant des explications précieuses tout au long de cette initiation, et me permettant ainsi de passer plus facilement et rapidement cette première étape pénible de familiarisation avec la formalisation, le langage et la bibliothèque.