

Arithmétique efficace des extensions de corps finis

Édouard Rousseau

▶ **To cite this version:**

Édouard Rousseau. Arithmétique efficace des extensions de corps finis. Number Theory [math.NT]. Institut Polytechnique de Paris, 2021. English. NNT : 2021IPPAT013 . tel-03299466

HAL Id: tel-03299466
https://theses.hal.science/tel-03299466

Submitted on 26 Jul 2021

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.

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2021IPPAT013

Thèse de doctorat

Efficient Arithmetic of Finite Field Extensions

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°574 École doctorale de mathématiques Hadamard (EDMH)
Spécialité de doctorat : Mathématiques fondamentales

Thèse présentée et soutenue à Versailles, le 12 Juillet 2021, par

M. EDOUARD ROUSSEAU

Composition du Jury :

Daniel Augot Directeur de recherche, INRIA Saclay (LIX)	Président
Stéphane Ballet Maître de conférence, Université d'Aix-Marseille (Equipe Arithmétique et Théorie de l'Information)	Rapporteur
Claus Fieker Professor, University of Kaiserslautern (Department of Mathematics)	Rapporteur
Julia Pieltant Maîtresse de conférence, CNAM (Équipe Sécurité et Défense)	Examinatrice
Luca De Feo Maître de conférence, IBM Research et Université de Versailles Saint-Quentin-en-Yveline (LMV)	Directeur de thèse
Eric Schost Professor, University of Waterloo (SCG)	Co-directeur de thèse
David Madore Maître de conférence, Télécom Paris (LTCI)	Invité
Hugues Randriambololona Maître de conférence, ANSSI et Télécom Paris (LTCI)	Invité

Contents

Préface **4**
Remerciements . . . 5
Applications des corps finis . . . 6
Cryptographie . . . 6
Théorie des codes . . . 10
Arithmétique des corps finis . . . 11
Résumé des travaux . . . 12
Arithmétique efficace dans une extension fixée . . . 13
Arithmétique efficace dans réseau d'extensions . . . 15

1 Introduction **18**
1.1 Finite fields in computer algebra . . . 19
1.2 Organization of the document . . . 20
1.2.1 Efficient arithmetic in a single finite field . . . 20
1.2.2 Efficient arithmetic in a lattice of finite fields . . . 22

I Efficient arithmetic in a single finite field **26**

2 Preliminaries **27**
2.1 Finite fields . . . 28
2.1.1 Finite field structure . . . 28
2.1.2 Subfields and field extensions . . . 28
2.2 Algebraic function fields . . . 30
2.2.1 Places . . . 31
2.2.2 Independence of valuations . . . 34
2.2.3 Divisors . . . 34
2.3 Complexity models . . . 37
2.3.1 Algebraic complexity . . . 38
2.3.2 Landau notations . . . 38
2.4 Fundamental algorithms . . . 39
2.4.1 Finite field arithmetic . . . 39
2.4.2 Classic routines . . . 40

3 Bilinear complexity and Chudnosky^2-type algorithms **41**
3.1 Bilinear complexity . . . 42
3.2 Chudnovsky-Chudnovsky algorithm . . . 46

3.2.1 Evaluation - Interpolation . . . 46
3.2.2 Asymptotic complexity . . . 48
3.3 Algorithmic searches in small dimension . . . 49
3.3.1 Barbulescu, Detrey, Estibals and Zimmerman's algorithm . . . 50

4 Hypersymmetric bilinear complexity 57
4.1 Symmetric and hypersymmetric fomulas . . . 58
4.1.1 Generalization to multilinear maps . . . 58
4.1.2 Trisymmetric and hypersymmetric complexity . . . 60
4.1.3 Galois invariance . . . 63
4.1.4 Multiplication formulas in algebras . . . 65
4.2 Algorithmic search in small dimension . . . 65
4.2.1 General algorithm description . . . 66
4.2.2 Implementation . . . 74
4.2.3 Universal formulas . . . 77
4.3 Asymptotic complexities . . . 80

II Efficient arithmetic in a lattice of finite fields 88

5 Isomorphism algorithms 89
5.1 Preliminaries and naive algorithm . . . 90
5.1.1 Description of the problem . . . 90
5.1.2 Embedding description problem and naive algorithm . . . 92
5.2 Lenstra-Allombert algorithm . . . 92
5.2.1 Preliminaries . . . 92
5.2.2 Kummer algebras . . . 96
5.2.3 The isomorphism algorithm . . . 101
5.2.4 Computing (H90) solutions . . . 102
5.3 The embedding evaluation problem . . . 103
5.3.1 Linear algebra . . . 104
5.3.2 Inverse maps and duality . . . 104
5.3.3 Modular composition . . . 108

6 From a single finite field to plenty: lattice of embeddings 110
6.1 The compatibility problem . . . 111
6.2 Conway polynomials . . . 112
6.3 The Bosma-Canon-Steel framework . . . 114
6.3.1 The Bosma-Canon-Steel algorithm . . . 114
6.3.2 Implementation in Nemo . . . 120

7 Standard lattice of compatibly embedded finite field 130
7.1 The Lenstra-Allombert algorithm and lattices of embeddings . . . 132
7.1.1 From isomorphism to embedding . . . 132
7.1.2 Cyclotomic lattices . . . 134
7.1.3 Kummer embeddings . . . 135
7.2 Standard solution of Hilbert 90 . . . 141
7.2.1 Complete algebras and standardization . . . 141

7.2.2 Towards standard embeddings . . . 146
7.3 Standard embeddings . . . 148
7.4 Implementation . . . 152
7.4.1 Complexity analysis . . . 153
7.4.2 Experimental results . . . 156

Conclusion **162**

Bibliography **163**

Préface

Cette préface est destinée à tous les lecteurs, toutes les lectrices, et pas seulement les mathématiciens et mathématiciennes. Après les remerciements d'usage, on présentera quelques applications des *corps finis*, l'objet au cœur de ce document, afin de comprendre l'étendue de leur utilité. Celles et ceux voulant assouvir leur soif de détails techniques et de mathématiques pourront le faire en parcourant les références bibliographiques proposées. Cela sera sans doute aussi possible (jusqu'à un certain point) en lisant les autres chapitres de ce manuscrit, ainsi que la fin de cette préface, qui donne un résumé détaillé du contenu du document.

Contents

Remerciements **5**
Applications des corps finis **6**
Cryptographie 6
Théorie des codes 10
Arithmétique des corps finis **11**
Résumé des travaux **12**
Arithmétique efficace dans une extension fixée 13
Arithmétique efficace dans réseau d'extensions 15

Remerciements

Je crois que ce document est le fruit d'innombrables rencontres et discussions, avec des personnes qui m'ont aidé, inspiré et enseigné (pas seulement en mathématiques), parfois sans qu'elles ne le sachent, et parfois sans que je ne le sache moi-même. Je n'arriverai donc pas à toutes les remercier ici, mais je ferai de mon mieux.

Mes premiers remerciements vont à mes directeurs de thèse Luca De Feo, Hugues Randriambololona et Eric Schost. Merci de m'avoir fait confiance, parfois quand on ne se connaissait pas beaucoup. Merci pour m'avoir fait découvrir de beaux sujets de recherche, pour votre disponibilité et vos conseils. Merci aussi pour l'ambiance de travail chaleureuse, cela a été un plaisir de faire des mathématiques avec vous.

Je voudrais également remercier les rapporteurs Stéphane Ballet et Claus Fieker, qui ont permis d'améliorer ce texte par leur travail de relecture minutieux et leurs remarques. Merci aussi à Daniel Augot, David Madore et Julia Pieltant de m'avoir fait l'honneur d'accepter de faire partie du jury de ma soutenance de thèse.

Je ne suis pas sûr que j'avais un amour prédestiné pour les mathématiques, je voudrais donc remercier tous ceux qui ont contribué à faire grandir mon goût pour cette discipline passionnante : tous mes enseignants depuis que je suis entré à l'école. En particulier, je remercie ceux qui m'ont encadré lors de stages ou de projets : Nicolas Thiéry, Stéphane Fischler, Pierre-Guy Plamondon et Michaël Quisquater.

J'ai été de passage dans plusieurs équipes pendant ma thèse, et malgré ma présence partielle dans chacune d'entre elles, j'y ai toujours trouvé des gens m'accueillant amicalement. Merci donc à tous les gens de passage, et bien entendu aux membres permanents, de l'équipe CRYPTO de l'université de Versailles, de l'équipe MC2 de Télécom Paris, et du *Symbolic Computation Group* de l'université de Waterloo. Vous êtes malheureusement trop nombreux pour que je ne commette pas un oubli impardonnable.

Merci à celles et ceux qui m'ont accompagné dans les autres moments : mes vieux copains de Cholet, et ceux rencontrés plus récemment à Orsay et Versailles. Merci aux Gibbons Masqués pour les moments d'évasion, de rire, mais aussi de débats sans fin lors des assemblées générales.

Enfin, merci à ma (belle-)famille pour son soutien indéfectible. On ne peut surévaluer tout ce que mes parents m'ont appris, et pour tout cela je leur serai éternellement reconnaissant. Et finalement merci à toi Amandine pour ta présence, pour le voyage qu'on a commencé et qu'on continue de mener, pour le bonheur quotidien et ineffable.

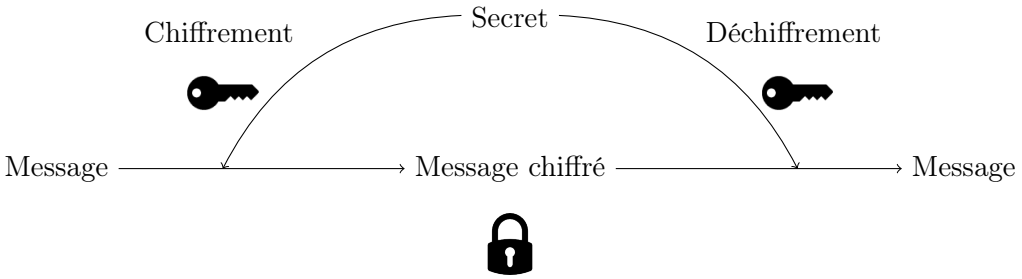


Figure 1: La stratégie générale d'un protocole de chiffrement symétrique.

Applications des corps finis

On peut se demander, probablement à juste titre, à quoi sert une thèse en mathématiques fondamentales. J'ai la chance d'avoir travaillé sur un sujet qui, quoique relativement abstrait, possède des applications extrêmement utiles, et ce dans la vie de tous les jours, pour quasiment tout le monde. Nous allons donc voir deux applications élégantes des *corps finis*, qui illustrent l'intérêt de ces objets mathématiques.

Cryptographie

Pendant toute la durée de mon doctorat, j'ai expliqué aux non-mathématiciens que je faisais une thèse en *cryptographie*. C'est en fait un mensonge, car même si le titre initial du projet de thèse était *Arithmétique efficace pour la cryptographie et la cryptanalyse*, je me suis finalement intéressé aux deux premiers mots seulement : arithmétique efficace. Cependant, la cryptographie reste une source d'inspiration et une des motivations derrière ces travaux. Les recherches que nous avons menées viennent souvent de la cryptographie, ou bien ont une application dans cette discipline, nous commencerons donc par expliquer ce que "cryptographie" signifie.

Nous sommes des animaux sociaux, et nous avons donc besoin de communiquer les uns avec les autres. Parfois, nous voulons que nos échanges restent privés. Les raisons derrière ce souhait peuvent être multiples : informations militaires, commerciales, médicales, bancaires, histoires amoureuses... La cryptographie est la science qui étudie les techniques utilisées pour sécuriser les communications, en présence d'une tierce partie appelée *adversaire*. Historiquement, la cryptographie s'est d'abord concentrée sur le chiffrement des messages (leur confidentialité), c'est-à-dire rendre le message illisible pour quelqu'un qui l'intercepterait ou en obtiendrait une copie. Pour que le destinataire légitime du message puisse le lire, il faut alors le *déchiffrer*. C'était uniquement possible lorsqu'à la fois l'émetteur du message et son destinataire partagaient un secret commun au préalable, secret qui était alors utilisé aussi bien pour chiffrer que pour déchiffrer le message. Dans un protocole cryptographique, le secret commun est appelé une *clé*, parce que le chiffrement est vu comme un cadenas. Cette méthode de chiffrement est appelée chiffrement symétrique car les deux participants partagent le même secret. La situation est résumée sur la Figure 1. Le chiffre de César est un vieil exemple de protocole cryptographique, dans lequel chaque lettre du message est remplacée par une autre lettre. Toutes les lettres sont décalées par un nombre constant n de positions vers le début de l'alphabet. Par exemple, avec $n = 3$, la lettre `D` devient `A`, la lettre `E` devient `B`, la lettre `F` devient `C`, et ainsi de suite. Ce protocole doit son nom à Jules César, qui l'utilisait pour communiquer avec sa famille, avec un décalage de $n = 3$. Dans la Figure 2, on a dessiné un schéma représentant la correspondance entre les

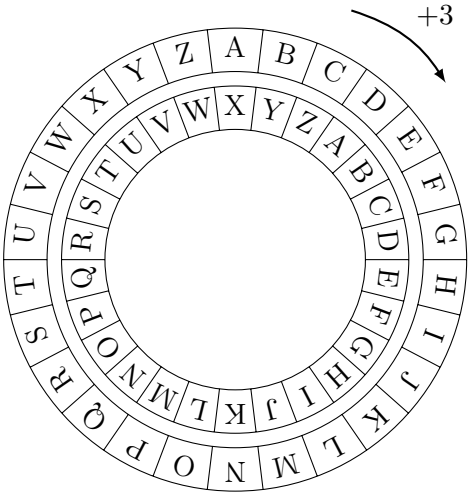


Figure 2: Representation du chiffre de César avec le décalage $n = 3$.

lettres avec le décalage $n = 3$. L'anneau extérieur représente les lettres dans le texte brut (le texte original, sans chiffrement), tandis que l'anneau intérieur correspond aux lettres dans le texte chiffré. Dans cet exemple, la clé secrète du protocole est la valeur n du décalage : connaissant n, on peut à la fois chiffrer et déchiffrer des messages. Le chiffre de César est suffisamment simple pour être exécuté par une machine, mais il n'est plus utilisé aujourd'hui. En effet, le faible nombre de clés possibles lorsque ce protocole est utilisé est petit, et un adversaire (un espion, un ennemi...) peut donc facilement deviner le message brut en essayant toutes les clés possibles. On pourrait même demander à un ordinateur de faire cette recherche, ce qui accélèrerait encore les choses. C'est pourquoi, en cryptographie moderne, le nombre de clés utilisables doit être bien plus grand. Par exemple, le protocole standard de chiffrement symétrique, appelé AES (pour Advanced Encryption Standard), a été créé en 1999 [DR99, DR02] et peut être utilisé avec 2^{128}, 2^{192}, ou 2^{256} clés différentes, en fonction de la version de chiffrement utilisée. Le plus petit de ces nombres peut aussi s'écrire

$$2^{128} = 340282366920938463463374607431768211456,$$

alors qu'un milliard s'écrit

$$10^9 = 1000000000,$$

ce sont donc des nombres vraiment très grands.

Le nombre de clés possibles n'est pas la seule chose ayant changé depuis Jules César. D'abord, les communications sont désormais essentiellement numériques, et donc la cryptographie fait partie de l'informatique. C'est très important parce que cela signifie que les travaux réalisés pendant cette thèse sont aussi orientés vers l'informatique : on souhaite obtenir des résultats mathématiques qui sont *effectifs*, c'est-à-dire qui sont utilisables par un ordinateur. Ensuite, l'étendue de la cryptographie est aujourd'hui bien plus large. Dans la cryptographie moderne, le chiffrement symétrique ne constitue qu'un seul domaine de la cryptographie, qui possède bien d'autres aspects, comme le chiffrement asymétrique (aussi appelé chiffrement à clé publique), l'intégrité des données, l'authentification, les signatures numériques (la liste n'est pas exhaustive). Nous n'expliquerons pas tous ces termes, mais les personnes intéressées peuvent lire les introductions sur chacun de ces sujets dans [MVOV18], par exemple. Un changement majeur en cryptographie a eu lieu en 1976, avec l'article pionnier *New Directions in Cryptography* [DH76] de Diffie et Hellman, qui ont

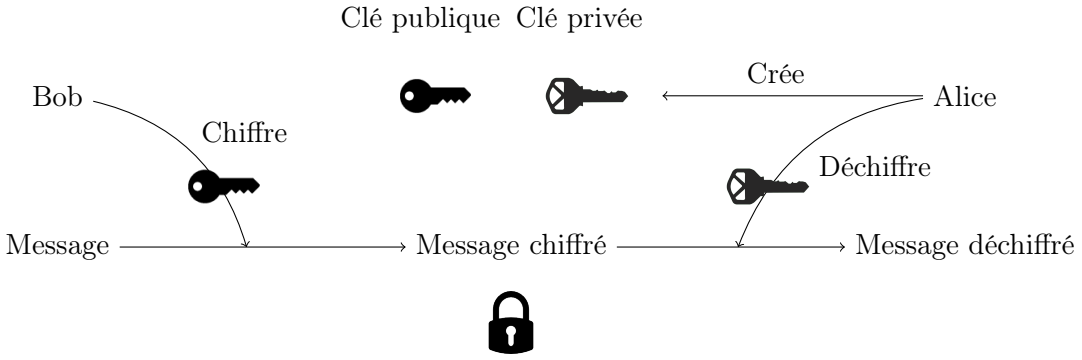


Figure 3: Idée générale du chiffrement à clé publique.

inventé ce qu'on appelle la cryptographie à clé publique. On présente brièvement la cryptographie à clé publique, afin de la comparer avec la cryptographie symétrique.

Un des désavantages principaux du chiffrement symétrique est que les deux personnes participantes doivent détenir un secret en commun afin de pouvoir communiquer de manière sécurisée. On peut imaginer qu'elles se donnent rendez-vous en personne pour se mettre d'accord sur un secret, mais cela n'est pas toujours possible, par exemple si elles vivent très loin l'une de l'autre. Elles pourraient alors trouver un autre moyen de communication pour s'échanger leur secret, mais ce moyen de communication ne serait pas sécurisé justement parce qu'elles n'ont pas encore pu échanger de secret. On a ainsi l'impression qu'on se mord la queue et que le problème est insoluble. En fait, le chiffrement à clé publique vient justement résoudre ce problème, car il permet de chiffrer des messages sans avoir connaissance d'un secret commun. L'idée très élégante de Diffie et Hellman est de casser la symétrie entre les personnes participantes (qu'on appellera Alice et Bob, car c'est la tradition en cryptographie). Au lieu de se mettre d'accord sur un secret commun, seul l'un des participants (par exemple Alice) crée une *paire* de clés : l'une d'elle va être publique et est destinée à être transmise à tout le monde, alors que l'autre est privée et doit être connue d'Alice seulement. Avec la clé publique, chacun peut chiffrer un message, alors que la clé privée est nécessaire quant à elle au déchiffrement. En utilisant ce genre de système, tout le monde peut envoyer un message chiffré à Alice, car la clé à utiliser pour cela est publique, mais seule Alice peut déchiffrer ces messages, même s'ils sont interceptés par un éventuel adversaire. Ainsi les communications restent sécurisées. Nous donnons un schéma du fonctionnement général du chiffrement à clé publique dans la Figure 3. Avec un tel système, Bob ne peut pas recevoir de message, il peut uniquement en envoyer à Alice. Si Alice veut envoyer un message à Bob en utilisant du chiffrement à clé publique, Bob doit alors créer sa propre paire de clé. Il donne alors sa clé publique à Alice, qui peut l'utiliser pour chiffrer un message et l'envoyer à Bob. Bob déchiffre alors le message en utilisant sa clé privée. La cryptographie à clé publique est plus lourde à mettre en place que la cryptographie symétrique, une autre solution est donc pour Bob de choisir un secret, de le chiffrer, puis de l'envoyer à Alice, qui pourra le déchiffrer et ils pourront alors tous les deux l'utiliser pour mettre en place un protocole de chiffrement symétrique. C'est ce qui est fait en pratique : seul un *échange de clé* a lieu en utilisant la cryptographie à clé publique, le reste étant géré par la cryptographie symétrique. Néanmoins, la cryptographie à clé publique est fondamentale car elle permet la mise en place de la cryptographie symétrique.

Le premier protocole d'échange de clé a été inventé par Diffie et Hellman en 1976 [DH76], et un exemple de chiffrement à clé publique est donné par Rivest, Shamir et Adleman avec le protocole RSA [RSA78] qu'ils ont décrit en 1977. Ces deux protocoles sont tous les deux basés

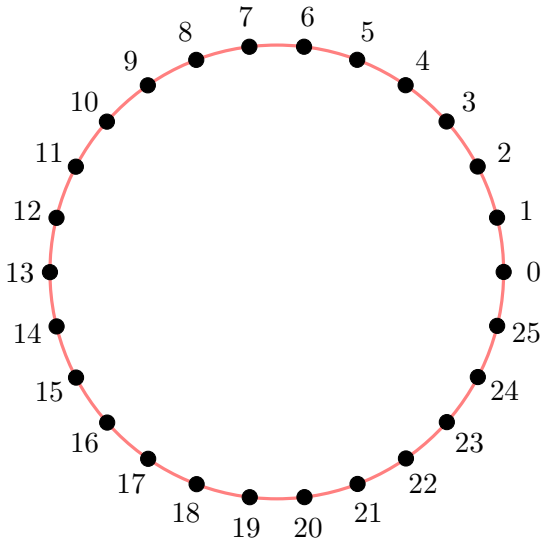


Figure 4: Le groupe cyclique $\mathbb{Z}/26\mathbb{Z}$ représenté par un cercle.

sur des structures mathématiques. En effet, les mathématiques sont un moyen pratique d'étudier et d'expliquer la cryptographie, par exemple le chiffre de César avec le décalage $n = 3$ peut être expliqué en représentant les lettres par des nombres entre 0 et 25

$$\texttt{A} \to 0, \texttt{B} \to 1, \ldots, \texttt{Y} \to 24, \texttt{Z} \to 25$$

et en définissant le chiffrement par la soustraction par 3. Avec cette représentation, on admet que le nombre -1 est équivalent au nombre 25, c'est-à-dire qu'avant le `A` vient la lettre `Z`, que le nombre -2 est équivalent au nombre 24, c'est-à-dire que deux lettres avant le `A` vient la lettre `Y`, et ainsi de suite. En fait, une partie des mathématiques appelée *théorie des nombres* est dédiée à l'étude de ce genre de nombres joints à des règles comme celle que nous venons d'énoncer : $-1 = 25$. Ces ensembles de nombres sont appelés des *groupes cycliques*, car ils peuvent être représentés par un cercle. Celui que nous avons évoqué est noté

$$\mathbb{Z}/26\mathbb{Z}$$

et peut être représenté par la Figure 4. De nombreuses autres structures intéressantes existent, on peut lire [Lan04, Per96] pour en apprendre plus à leur sujet ou pour découvrir les richesses de l'algèbre. Avec le développement de la cryptographie à clé publique, la théorie des nombres en est devenue la pierre angulaire, et des concepts mathématiques plus élaborés ont été utilisés, comme les corps finis, les courbes elliptiques, ou les isogénies. Sans entrer dans les détails, ce qui est important est que la sécurité des protocoles cryptographiques est basée sur des problèmes mathématiques difficiles faisant intervenir ces concepts. Ainsi, une meilleure compréhension de ces objets implique une meilleure compréhension de la sécurité de nos protocoles cryptographiques. La partie de la cryptographie dont le rôle est d'analyser la sécurité de ces protocoles (c'est-à-dire, de les "casser") est appelée la *cryptanalyse*.

En outre, puisque les protocoles sont basés sur des manipulations dans ces objets, une meilleure compréhension de ces derniers implique également de meilleurs (en particulier, plus rapides) protocoles cryptographiques. Comme la cryptographie est omniprésente dans les communications modernes (sur Internet, quand vous utilisez votre carte de crédit, sur les applications de messagerie

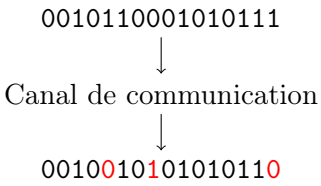


Figure 5: Un canal de communication imparfait.

de votre téléphone, ...), avoir des protocoles efficaces est crucial. Il est donc nécessaire d'être capable de manipuler efficacement les concepts mathématiques qui se cachent derrière nos protocoles, que ce soit pour la cryptographie (mettre en place des protocoles performants) ou la cryptanalyse (analyser leur sécurité). Par "manipuler", on entend être capable de faire des additions, des multiplications, et parfois des opérations plus compliquées avec ces objets. La science qui étudie comment faire tout cela s'appelle *l'arithmétique*. En conclusion, il est nécessaire d'avoir une arithmétique efficace pour la cryptographie et la cryptanalyse.

Théorie des codes

Une autre application élégante (et utile) des corps finis est la *théorie des codes*. Comme la cryptographie, la théorie des codes est liée aux communications, mais elle répond à un problème complètement différent. Nous faisons maintenant l'hypothèse qu'Alice et Bob veulent communiquer, et que les informations qu'ils veulent échanger vont traverser un canal de communication. En pratique, ce canal peut être de la fibre optique, des fils électriques, des ondes radio, et beaucoup d'autres choses encore. On suppose qu'Alice veut envoyer le message "Salut ! Ça va ?" à Bob. En réalité, ces lettres vont probablement d'abord être transformées en une suite de bits : `0` et `1`, on imaginera donc qu'Alice envoit le message

$$m = \texttt{0010110001010111}$$

à travers le canal. Souvent, dans des conditions réelles, le canal de communication est imparfait, et est sujet à du *bruit*, c'est-à-dire qu'à cause de la qualité du canal, ou à cause de l'environnement, des erreurs peuvent apparaître et changer le message envoyé par Alice, comme montré sur la Figure 5. Sans autre outil ou contexte, il peut être difficile (voire impossible) pour Bob de deviner ce que voulait dire Alice. La théorie des codes étudie les moyens de lutter contre l'apparition d'erreurs et de retrouver le message initial, même une fois modifié. Le but est de construire ce qu'on appelle des *codes* qui vont être capable de corriger des erreurs, on les nommera donc *codes correcteurs d'erreurs*. L'idée est que pour être sûr qu'une information, qui est représentée par un `0` ou un `1`, arrive jusqu'à Bob sans erreur, Alice va la répéter plusieurs fois. Par exemple, si Alice répète chaque bit trois fois, en faisant des mots de trois symboles :

$$\texttt{0} \longrightarrow \texttt{000} \qquad \texttt{1} \longrightarrow \texttt{111},$$

alors une erreur peut être corrigée, car si Bob reçoit, par exemple, le mot `001`, il sait qu'il s'agit probablement du mot `000` qui a subit une erreur de transmission. Il peut alors interpréter le mot `001` comme venant du bit initial `0` dans le message d'Alice. Plus généralement, si Bob reçoit un mot `abc`, il l'interprétera comme venant du bit `0` s'il y a une majorité de `0` dans le mot `abc`. Réciproquement, s'il y a une majorité de `1` dans le mot `abc`, il l'interprétera comme venant du bit `1`. Bien entendu, s'il y a trop d'erreurs de transmission dans un seul mot, il est possible de mal

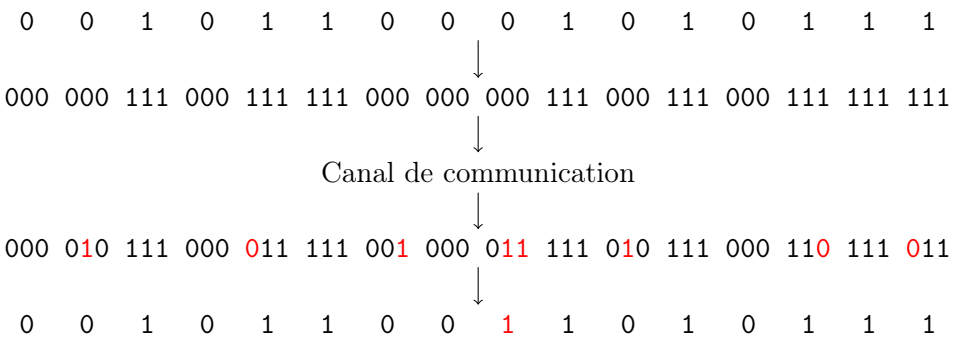


Figure 6: Le code de répétition de longueur 3.

interpréter le message. Par exemple, si le mot `000` devient `110` après être passé dans le canal de communication, alors Bob le décodera comme `111` $\rightarrow$ `1`. Cette stratégie est connue sous le nom de code de répétition de longueur 3, et est un exemple de code correcteur d'erreur. La procédure complète est décrite dans la Figure 6. Afin de corriger plus d'erreurs, on pourrait utiliser des codes de répétition avec des longueurs plus grandes. Néanmoins, lorsqu'on utilise un code de répétition de longueur n, il faut envoyer n fois plus d'information à travers le canal, ce qui peut avoir un prix dans des situations réelles. Il faut donc trouver un équilibre entre le surcoût induit par la longueur du code et le nombre d'erreurs que l'on souhaite être capable de corriger. En fait, en fonction de la qualité du canal de communication, tous les codes ne conviennent pas. Claude Shannon a prouvé en 1948 [Sha48] que l'on peut définir une quantité appelé *capacité* du canal qui mesure essentiellement la quantité d'information que l'on peut faire passer à travers un canal. Cela mesure aussi combien de répétitions on doit inclure dans notre code pour que le message soit transmis de manière fiable. La théorie des codes est un domaine de recherche actif, et il n'existe donc pas de réponse définitive à la question "Quel est le meilleur code que nous pouvons utiliser ?". Les codes utilisés en pratique sont un peu plus subtils que le code de répétition, mais ils suivent la même logique. Par exemple, les codes de Reed-Solomon [RS60], utilisés dans des technologies de la vie de tous les jours comme les CDs, les DVDs, les disques Blu-ray, ou encore dans les missions de la NASA, font intervenir des mots composés de suites de valeurs prises dans un corps fini. D'autres codes, comme les codes de Goppa géométriques [Gop81], sont basés sur les corps de fonctions algébriques, une structure mathématique qui est construite à partir des corps finis. En conclusion, exactement comme en cryptographie, les corps finis sont omniprésents en théorie des codes, et une arithmétique efficace dans les corps finis permet à la fois un codage et un décodage performant.

Arithmétique des corps finis

Comme énoncé dans les pages précédentes, les protocoles cryptographiques et les codes sont basés sur des structures mathématiques pour fonctionner. Celle que nous étudions tout au long de ce document est appelée *corps fini*. Un *corps* est une structure mathématique (on dit aussi *structure algébrique*) composée d'éléments que l'on peut additionner, soustraire, multiplier, et diviser (excepté par zéro). C'est une structure bien connue : l'ensemble des nombres réels, noté $\mathbb{R}$, en est un exemple. En effet, les éléments de $\mathbb{R}$, par exemple 0, 1, $-3,5631$, mais aussi des nombres plus compliqués comme π, $\sqrt{2}$, ou $\frac{7}{13}$ peuvent être additionnés, soustraits, multipliés, ou divisés. Les nombres dans $\mathbb{R}$ peuvent avoir une infinité de décimales, par exemple les 60 premières

décimales de la constante π sont

$$\pi = 3.14159265358979323846264338327950288419716939937510582097494\ldots.$$

Un ordinateur n'a qu'une mémoire finie, c'est-à-dire que la quantité d'information qu'il peut stocker n'est pas illimitée. Par conséquent, il est impossible de stocker toutes les décimales de π, et, plus généralement, les décimales de beaucoup d'autres nombres de $\mathbb{R}$, sur un ordinateur. Cela reste possible de travailler avec des nombres dans $\mathbb{R}$ sur un ordinateur, mais il est plus simple de manipuler des éléments qui viennent d'une structure algébrique *finie*, c'est-à-dire composée d'un nombre n d'éléments, avec

$$n < \infty.$$

Un exemple de ce type de structure a déjà été donné : le groupe cyclique $\mathbb{Z}/26\mathbb{Z}$ qui est composé des "nombres" $0, 1, \ldots, 25$. Cependant, ce ne sont pas les mêmes nombres que ceux que nous connaissons, car avec les éléments de $\mathbb{Z}/26\mathbb{Z}$, on a par exemple l'égalité

$$3 - 4 = -1 = 25,$$

ce qui n'est pas correct pour les vrais nombres dans $\mathbb{R}$, mais nous écrivons tout de même les nombres de $\mathbb{Z}/26\mathbb{Z}$ comme ceux de $\mathbb{R}$ par simplicité. Nous avons déjà défini une addition et une soustration dans $\mathbb{Z}/26\mathbb{Z}$, et nous pourrions aussi définir une multiplication et une division d'une manière naturelle. Les corps finis sont une généralisation des espaces comme $\mathbb{Z}/26\mathbb{Z}$. Bien que l'ensemble $\mathbb{Z}/26\mathbb{Z}$ ne soit pas un corps, pour des raisons techniques, cela reste une bonne approximation, pour cette introduction en tous cas, de penser les corps finis comme ce genre d'ensemble.

Comme les éléments d'un corps fini sont, par définition, en nombre fini, c'est relativement plus facile de les manipuler avec un ordinateur. De plus, la structure de corps nous permet de manipuler les éléments des corps finis comme des nombres classiques (c'est-à-dire avec des additions, multiplications, ...), ce qui les rend utiles. C'est pourquoi aujourd'hui, les corps finis sont absolument partout dans des domaines comme la cryptographie ou la théorie des codes.

Parfois, certains problèmes mathématiques simples sont bien compris d'un point de vue théorique, mais il existe toujours des questions ouvertes concernant certains aspects pratiques. Par exemple, la multiplication de deux entiers a et b de $\mathbb{N}$ est un problème simple, qui peut être fait à la main par des enfants. Pourtant, la façon optimale de multiplier deux entiers grâce à un ordinateur reste un problème ouvert, pour lequel des articles de recherche sont régulièrement publiés [HVDH19a]. L'arithmétique des corps finis, c'est-à-dire comment faire des opérations comme l'addition ou la multiplication, est très bien comprise, car la structure des corps finis est assez simple. Pour autant, la meilleure façon de multiplier deux éléments dans un corps fini est également inconnue. C'est exactement le sujet de cette thèse : l'étude de l'arithmétique des corps finis.

Résumé des travaux

Ce résumé présente de manière détaillée l'ensemble des travaux réalisés pendant ma thèse, ainsi que la façon dont est organisé ce manuscrit. Cette partie est donc dédiée à un "public scientifique averti".

Arithmétique efficace dans une extension fixée

Ce document est composé de deux parties, qui sont essentiellement indépendantes. Dans la Partie I, nous étudions l'arithmétique d'une extension de corps fini fixée

$$\mathbb{F}_{p^k}.$$

Préliminaires. Nous commençons dans le Chapitre 2 par rappeler les faits fondamentaux concernant les objets que nous utiliserons dans le reste du document. En particulier, on rappelle la structure des corps finis, ainsi que la manière de les construire comme des quotients d'anneaux de polynômes

$$\mathbb{F}_p[x]/(P(x)).$$

Nous donnons quelques propriétés des extensions de corps finis concernant leur structure d'espace vectoriel ainsi que sur leur groupe d'automorphismes.

Dans la Section 2.2, nous présentons les *corps de fonctions algébriques*, une structure algébrique que nous utilisons dans les preuves des Chapitres 3 et 4. On rappelle brièvement ce que sont les *places*, et qu'elles sont essentiellement équivalentes aux notions de valuations discrètes et d'anneaux de valuations. On décrit comment évaluer un élément z d'un corps de fonction algébrique à une place P, et on donne la définition d'un zéro et d'un pôle, ce qui justifie le nom de corps de "fonction". Enfin, on donne la définition d'un *diviseur* et on rappelle les résultats habituels de la théorie : le lien entre le degré et la dimension d'un diviseur, la définition du *genre* d'un corps de fonctions algébrique, et le théorème de Riemann-Roch.

Dans la Section 2.3, on donne le modèle de la *complexité algébrique* et on explique pourquoi ce modèle est pertinent pour nos travaux. On rappelle également les notions de *grand O* et *petit o*, qui sont utilisées pour exprimer des résultats asymptotiques.

Enfin, dans la Section 2.4, nous donnons des références bibliographiques, ainsi que la complexité de certains algorithmes classiques et importants dans notre travail : l'algorithme de Brent-Kung pour la composition modulaire, le calcul de polynôme minimal, et l'algorithme de Berlekamp-Massey. Ces routines sont utilisées dans plusieurs algorithmes de ce manuscrit, notamment dans les Chapitres 5 et 7.

Complexité bilinéaire et algorithmes de type Chudnovsky^2. Dans le Chapitre 3, nous présentons la théorie de la *complexité bilinéaire*, un modèle alternatif de complexité utilisé pour mesurer le coût de calcul de certaines applications bilinéaires. Dans ce modèle, seules les multiplications sont comptées, ce qui est justifié par le fait qu'en pratique les multiplications sont plus coûteuses que les additions. La complexité bilinéaire d'une application est alors donnée par le nombre minimal de multiplications nécessaires au calcul de l'application en question. L'algorithme de Karatsuba est un exemple pratique de l'intérêt de ce modèle de complexité. En effet, l'idée derrière cet algorithme est de multiplier deux polynômes de degré 1

$$A = a_1X + a_0 \text{ et } B = b_1X + b_0$$

avec seulement trois produits

$$c_0 = a_0b_0,$$

$$c_1 = (a_0 + a_1)(b_0 + b_1),$$

et

$$c_\infty = a_1b_1,$$

à la place des quatre produits classiques a_0b_0, a_0b_1, a_1b_0 et a_1b_1 comme suit :

$$AB = c_\infty X^2 + (c_1 - c_\infty - c_0)X + c_0.$$

Dans la Section 3.1, après avoir rigoureusement défini la complexité bilinéaire d'une application bilinéaire Φ en utilisant des *formules bilinéaires*, nous expliquons que cette définition est en réalité équivalente au rang du tenseur correspondant à Φ. Nous présentons également la version *symétrique* de la complexité bilinéaire, qui compte le nombre minimal de multiplications symétriques nécessaires au calcul de Φ, et qui est également étudiée dans la littérature. Nous sommes particulièrement intéressés par la multiplication dans les extensions de corps finis, et par sa complexité bilinéaire (symétrique).

Dans la Section 3.2, nous redonnons le principe d'évaluation-interpolation et nous présentons un algorithme dû à Chudnovsky et Chudnovsky [CC88] basé sur l'évaluation et l'interpolation sur des places d'un corps de fonctions algébrique. Cet algorithme fondateur donne une borne asymptotique *linéaire* en le degré de l'extension, que ce soit pour la complexité bilinéaire ou la complexité bilinéaire symétrique de la multiplication dans une extension de corps fini. Il a été très étudié, et nous présentons brièvement quelques améliorations [BR04, CÖ10, Ran12].

Toutefois, l'algorithme de Chudnovsky et Chudnovsky donne uniquement une borne sur la complexité bilinéaire, qui s'avère être asymptotiquement bonne. Ainsi, nous donnons aussi dans la Section 3.3 un algorithme de Barbulescu, Detrey, Estibal et Zimmermann [BDEZ12] qui permet de calculer la complexité bilinéaire en petite dimension. Leur algorithme énumère toutes les formules bilinéaires pour une longueur donnée, et peut ainsi être utilisé pour obtenir la longueur minimale d'une formule permettant de calculer une application bilinéaire Φ, autrement dit cet algorithme permet de calculer la complexité bilinéaire de Φ. L'algorithme est expliqué en détails, et quelques améliorations dûes à Covanov [Cov19] sont mentionnées.

Complexité bilinéaire hypersymétrique. Dans le Chapitre 4, nous nous intéressons à de nouveaux modèles de complexité, et nous donnons des résultats à la fois en petite dimension et asymptotiques. Dans la Section 4.1, nous généralisons la notion de complexité bilinéaire dans le cas du produit de $s \geq 2$ variables

$$x_1 \times x_2 \times \ldots x_{s-1} \times x_s$$

dans une extension de corps fini $\mathbb{F}_{p^k}$. En adaptant l'algorithme de Chudnovsky et Chudnovsky dans ce cas, nous montrons dans la Section 4.3 que cette *complexité multilinéaire* reste linéaire en le degré k de l'extension, comme dans le cas de la complexité bilinéaire classique.

Dans ce chapitre, on définit également une nouvelle complexité appelée *complexité bilinéaire hypersymétrique*, qui est inspirée de la complexité bilinéaire symétrique usuelle, dans laquelle une condition de symétrie additionnelle est étudiée. Dans la Section 4.2, nous donnons un algorithme *ad hoc* de recherche de formules hypersymétriques, inspiré par l'algorithme de Barbulescu, Detrey, Estibal and Zimmermann, qui nous permet de calculer la complexité bilinéaire hypersymétrique. Nous analysons notre algorithme en détails, et donnons des résultats expérimentaux issus de notre implémentation, dans le cas de la multiplication dans les extensions de corps finis. En utilisant des *formules universelles*, c'est-à-dire des formules qui sont vraies pour presque tout nombre premier p, nous donnons aussi des résultats théoriques concernant la complexité bilinéaire hypersymétrique dans les extensions de corps finis

$$\mathbb{F}_{p^k}$$

et dans des algèbres de polynômes tronqués

$$\mathbb{F}_p[T]/(T^k)$$

en petite dimension k, qui généralisent des résultats connus pour la complexité bilinéaire. Nous obtenons également la linéarité de la complexité bilinéaire hypersymétrique de la multiplication dans l'extension $\mathbb{F}_{p^k}$ en le degré de l'extension k, comme corollaire du même résultat pour la complexité multilinéaire.

Arithmétique efficace dans un réseau d'extensions

Dans la Partie II, nous étudions comment gérer plusieurs corps finis simultanément, dans ce que l'on appelle un *réseau de corps finis compatiblement plongés*. D'un point de vue théorique, cela revient à se demander comment calculer dans la clôture algébrique

$$\bar{\mathbb{F}}_p = \bigcup_{k \geq 1} \mathbb{F}_{p^k}$$

du corps de base $\mathbb{F}_p$.

Algorithmes d'isomorphisme. Le Chapitre 5 est dédié au problème de l'isomorphisme, qui demande de calculer efficacement un isomorphisme (ou plus généralement un plongement)

$$K \hookrightarrow L$$

entre deux corps finis K et L. Dans la Section 5.1, nous exposons le problème de l'isomorphisme, et, suivant la présentation faite dans [BDFD+17], nous le divisons en deux parties : le problème de la description du plongement et le problème de l'évaluation du plongement. Le problème de la description du plongement consiste à trouver des éléments $\alpha \in K$ et $\beta \in L$ tels que

$$K = \mathbb{F}_p(\alpha)$$

et tel qu'il existe un plongement $\phi : K \hookrightarrow L$ qui envoie α vers β. Connaissant α et β, le problème de l'évaluation du plongement consiste alors à évaluer ϕ de manière efficace. Nous traitons d'abord le problème de la description et nous présentons l'algorithme naïf, basé sur la factorisation de polynômes, dans la Section 5.1.2.

Dans la Section 5.2, nous présentons un algorithme plus élaboré dû à Allombert [All02a] et inspiré par le travail de Lenstra [LJ91], que nous appelons algorithme de Lenstra-Allombert. L'algorithme de Lenstra-Allombert est basé sur la théorie de Kummer, qui étudie certaines extensions de corps, et utilise des racines primitives de l'unité. Si l'extension $\mathbb{F}_{p^n}$ admet une racine n-ième de l'unité primitive ζ_n, alors la description de l'algorithme est plus simple et est donnée dans la Section 5.2.1. Sinon, il est nécessaire d'ajouter artificiellement une racine ζ_n à l'extension $\mathbb{F}_{p^n}$, ce qui nous conduit à étudier les algèbres de la forme

$$A_n = \mathbb{F}_{p^n} \otimes \mathbb{F}_p(\zeta_n)$$

que nous appelons *algèbres de Kummer*. Les éléments α et β décrivant le plongement donné par l'algorithme de Lenstra-Allombert sont alors déduits des solutions d'équations de la forme

$$(\sigma \otimes 1)(x) = (1 \otimes \zeta)x$$

dans des algèbres de Kummer. Ces équations sont appelées Hilbert 90. Les algèbres de Kummer, ainsi que les solutions de Hilbert 90, sont étudiées en détail dans la Section 5.2.2, car elles jouent un rôle central dans le Chapitre 7.

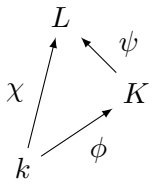
En utilisant les résultats de la Section 5.2.2, nous expliquons dans la Section 5.2.3 comment obtenir les éléments α et β des solutions de Hilbert 90. Puis, on présente les techniques pour obtenir les solutions de Hilbert 90 dans la Section 5.2.4. Enfin, les techniques standards permettant de répondre au problème de l'évaluation du plongement sont traitées dans la Section 5.3.

D'un seul corps fini vers une multitude : réseaux de plongements. Dans le Chapitre 6, nous étudions *le problème de la compatibilité*, qui demande comment calculer efficacement des plongements entre potentiellement beaucoup plus que deux corps finis, de manière compatible, c'est-à-dire tel que les diagrammes issus des plongements commutent. En d'autres termes, dès que l'on a trois extensions de corps finis

$$k \subset K \subset L$$

et des plongements $\phi : k \to K$, $\psi : K \to L$ et $\chi : k \to L$ entre eux, nous voulons l'égalité

$$\chi = \psi \circ \phi.$$



Le problème général, ainsi que des objectifs additionnels, sont présentés dans la Section 6.1.

Dans la Section 6.2, nous donnons une première solution, basée sur les polynômes de Conway [Par, Sch92]. Ces polynômes sont utilisés pour définir les extensions de corps finis et possèdent une propriété très intéressante de *compatibilité aux normes*, c'est-à-dire que prendre la norme d'une racine d'un polynôme de Conway donne une racine d'un autre polynôme de Conway. Cela donne des plongements faciles à calculer : en effet, le problème de la description du plongement est résolu en calculant une norme. Naturellement, avant d'utiliser les polynômes de Conway, il faut d'abord les calculer. Ceci constitue un problème car nous ne connaissons pas d'algorithme efficace pour calculer les polynômes de Conway. Par conséquent, ils sont le plus souvent précalculés jusqu'à un certain degré d dans la plupart des systèmes de calcul formel, et il n'est plus possible, ou alors très coûteux, de calculer des plongements avec des extensions de corps fini qui ont un degré supérieur à d.

Dans la Section 6.3, nous présentons une autre solution au problème de la compatibilité appelé l'algorithme de Bosma-Canon-Steel [BCS97]. Cet algorithme permet l'utilisation de polynômes arbitraires pour définir nos extensions de corps finis, et est *incrémentale*, c'est-à-dire qu'on peut ajouter de nouvelles extensions dans notre structure de données sans avoir à recalculer quoi que ce soit. Cette solution est beaucoup plus souple que celle donnée par les polynômes de Conway, et est donc une alternative très intéressante. On décrit l'algorithme en détails dans la Section 6.3.1. Cet algorithme a d'abord été disponible au sein du système de calcul formel MAGMA [BCP97] uniquement. À ma connaissance, le seul autre logiciel utilisant l'algorithme de Bosma-Canon-Steel est Nemo [H+16]. En effet, il a été implémenté à l'Automne 2019, lorsque j'ai gentiment été invité à l'université de Kaiserslautern par les développeurs de Nemo. Tous les détails de l'implémentation, et les résultats expérimentaux, sont donnés dans la Section 6.3.2.

Réseau standard de corps finis compatiblement plongés. Enfin, dans le Chapitre 7, nous construisons une nouvelle méthode [DFRR19] pour calculer des réseaux de corps finis compatiblement plongés, qui est à mi-chemin entre les polynômes de Conway et l'algorithme de Bosma-Canon-Steel. L'élément central de cette construction est l'algorithme de plongement de Lenstra-Allombert. Dans la Section 7.1, nous expliquons comment cet algorithme peut être utilisé pour produire des plongements compatibles, en choisissant des solutions de Hilbert 90 compatibles, et les nouveaux défis que cela engendre.

Dans la Section 7.2, nous montrons que certaines grosses algèbres de Kummer appelées *algèbres de Kummer complètes*, qui sont décrites par

$$A_{p^a-1} = \mathbb{F}_{p^{p^a-1}} \otimes \mathbb{F}_p(\zeta_{p^a-1}) = \mathbb{F}_{p^{p^a-1}} \otimes \mathbb{F}_{p^a},$$

admettent des solutions spéciales de Hilbert 90, que l'on nomme des *solutions standards*. Nous prouvons ensuite que de ces solutions standards dans l'algèbre complète A_{p^a-1}, nous pouvons déduire des solutions de Hilbert 90 dans n'importe quelle algèbre de Kummer

$$A_n = \mathbb{F}_{p^n} \otimes \mathbb{F}_p(\zeta_n)$$

telle que $[\mathbb{F}_p(\zeta_n) : \mathbb{F}_p] = a$. Nous remarquons que les solutions déduites des solutions standards partagent elles aussi des propriétés remarquables, et que toutes ces solutions peuvent être utilisées pour construire des plongements compatibles entre les corps finis $\mathbb{F}_{p^n}$ qui leur sont associés.

Dans la Section 7.2.2, nous montrons que si $a \mid b$, les solutions de Hilbert 90 dans les algèbres de Kummer complètes A_{p^a-1} et A_{p^b-1} peuvent aussi être reliées via un opérateur que l'on appelle *opérateur de norme scalaire* et qui agit essentiellement comme la norme classique des extensions de corps finis. Nous utilisons ce résultat dans la Section 7.3 pour obtenir des plongements compatibles entre deux extensions arbitraires $\mathbb{F}_{p^m}$ et $\mathbb{F}_{p^n}$, dès lors que l'on a $m \mid n$.

Nous analysons la complexité de notre nouvelle construction dans la Partie 7.4 et nous donnons une implémentation dans le language de programmation Julia [Jul], en utilisant Nemo. Nous voyons que notre construction peut être utilisée en pratique, et que les plongements peuvent être calculés en temps raisonnable.

Chapter 1

Introduction

In this Chapter, we briefly recall the special role of finite fields in computer algebra. We explain some challenges, as well as our contributions in understanding them. We also give an overview of the whole document and specify what to expect in each chapter.

Contents

1.1 Finite fields in computer algebra 19

1.2 Organization of the document 20

1.2.1 Efficient arithmetic in a single finite field 20

1.2.2 Efficient arithmetic in a lattice of finite fields 22

1.1 Finite fields in computer algebra

Finite fields are the central object of this thesis. They play a special role in computer algebra, since they are the building block of more complex structures like for example vector spaces of matrices or of polynomials. Other useful objects, like agebraic curves, are also built upon them. As a consequence, the applications of finite fields are abundant. They are especially important in modern communication, since they are ubiquitous in coding theory and cryptography. Because of their special role, it is crucial to understand how to efficiently compute in finite fields, and there has been extensive research on how to do so. Let

$$\mathbf{k} = \mathbb{F}_p$$

be a finite field of size p and let

$$\mathbb{F}_{p^k}$$

be an extension of $\mathbf{k}$ of degree k. There are several ways of representing the extension $\mathbb{F}_{p^k}$, and a very common one is to represent the elements of $\mathbb{F}_{p^k}$ by polynomials over $\mathbf{k}$ modulo some irreducible polynomial of degree k, *i.e.* we construct $\mathbb{F}_{p^k}$ as

$$\mathbb{F}_{p^k} = \mathbf{k}[T]/(P(T)),$$

where $P \in \mathbf{k}[T]$ is an irreducible polynomial of degree k. With this representation, addition between two elements x and y in $\mathbb{F}_{p^k}$ is done by adding the coefficients of the polynomials representing x and y, thus addition in $\mathbb{F}_{p^k}$ costs up to k additions in $\mathbf{k}$. The situation for multiplication is not so clear, since it is linked with polynomial multiplication, which is a more complex operation. The optimal way of multiplying two polynomials over a finite field is not known, and there was even some progress on that question quite recently [HVDH19b], yielding an algorithm for finite field multiplication in $\mathbb{F}_{p^k}$ with an asymptotic complexity of $O(k\log(k))$. In practice, depending on the size of k, the fastest way of multiplying elements in $\mathbb{F}_{p^k}$ is different, thus computer algebra systems typically change which algorithm is used depending on the size of k. For very small k, the naive multiplication algorithm for polynomials, that has an asymptotic (algebraic) complexity of $O(k^2)$, is the best. Then Karatsuba's algorithm [Kar63] is used, with a complexity of $O(k^{\log_2(3)})$ and $\log_2(3) \approx 1.58$. Finally the best option for large k is to use the Fast Fourier Transform (FFT) [CT65, SS71], with a complexity of $O(k\log(k)\log\log(k))$.

All these costs are measured using the algebraic complexity model, where all operations are assumed to cost the same unit amount. However, as suggested by timings in practice, the multiplication operation in $\mathbf{k}$ is more expensive (*i.e.* takes more time) than addition. Thus, in order to obtain efficient multiplication algorithms in finite field extensions $\mathbb{F}_{p^k}$, research has been done on the minimal number of multiplications in $\mathbf{k}$ needed to multiply two elements in $\mathbb{F}_{p^k}$. This quantity, known as the *bilinear complexity*, is hard to compute in general and is only known for small extensions. Even when it is known that two elements in $\mathbb{F}_{p^k}$ can be multiplied using n multiplications in $\mathbf{k}$, actual formulas using n multiplications are not always known. There is still a lot to discover in bilinear complexity theory.

Since the cost of addition is linear in the extension degree, and the cost of multiplication is quasi-linear, computer algebra systems handle finite field extensions arithmetic quite well. The links between finite field extensions are well understood, *e.g.* we know that

$$\mathbb{F}_{p^k} \subset \mathbb{F}_{p^l}$$

if and only if $k \mid l$, we know how to construct extensions

$$\mathbb{F}_{q^k}/\mathbb{F}_q$$

over a non-prime field $\mathbb{F}_q$, with $q = p^l$ and $l > 1$. Nevertheless, these links are often not handled by computer algebra system, *e.g.* the number theory library FLINT [Har10] only supports extensions of prime fields. In Sagemath [Dev21], the same is true for default finite fields, where asking for an extension of degree 3 of $\mathbb{F}_{p^3}$ gives the extension

$$\mathbb{F}_{p^9}/\mathbb{F}_p,$$

that is described by an irreducible polynomial in $\mathbf{k}[x]$ of degree 9. In the majority of computer algebra systems, it is impossible to easily manage user-defined finite fields, *i.e.* compatible embeddings between finite fields are supported only when the fields are defined using Conway polynomials. Consequently, even if the arithmetic of each finite field extension is usually efficient, there is often room to improvement when it comes to the management of several finite field extensions.

Yet, these features are sometimes needed in algorithms, for example the quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic [GKZ14] is based on a tower of extensions of $\mathbb{F}_q$, and deals with polynomials defined over a field K that changes during the algorithm and belongs to the tower.

Contributions. While preparing this thesis, we studied both the arithmetic of individual finite field extensions, and the arithmetic of several extensions at once. In [RR21], we generalize results known for the bilinear complexity to the *multilinear complexity* and to the *hypersymmetric complexity*. We also give a new algorithm to find formulas for the multiplications in $\mathbf{k}$-algebras, and give experimental results in the case of a finite field extension $\mathbb{F}_{p^k}$.

Concerning the arithmetic of several extensions, we studied how to deal with an arbitrary collection of extension, that we call a *lattice of compatibly embedded finite fields*. In [DFRR18], we describe an implementation of a framework due to Bosma, Canon and Steel [BCS97], that allows to embed user-defined finite fields, and is thus an alternative to Conway polynomials. Thanks to the kind invitation by its developpers, this implementation is now part of the computer algebra system Nemo. Prior to that, it was only available in Magma [BCP97]. We also worked on a new framework for computing compatible embeddings between finite fields in [DFRR19]. This new framework is an in-between of the Bosma-Canon-Steel framework and the Conway polynomials and brings new ideas to compute compatible embeddings, based on Kummer theory.

This document also presents an extended version of the articles [RR21] and [DFRR18], respectively in Parts I and II.

1.2 Organization of the document

1.2.1 Efficient arithmetic in a single finite field

This work is composed of two parts, that are essentially independent. In Part I, we study the arithmetic of one fixed finite field extension

$$\mathbb{F}_{p^k}.$$

Preliminaries. We begin in Chapter 2 by recalling fundamental facts about the mathematical objects that we use in the rest of the document. We present some properties about *finite fields*, that are at the center of this thesis. In particular, we recall the structure of finite fields, and how

to construct them as quotients of polynomial rings

$$\mathbb{F}_p[x]/(P(x)).$$

We also give some properties of finite field extensions concerning their vector space structure and their group of automorphisms.

In Section 2.2, we present *algebraic function fields*, an algebraic structure that we use in some proofs in Chapters 3 and 4. We briefly recall what *places* are, and that they are essentially equivalent to a discrete valuation or a valuation ring. We describe how to evaluate an element z of the algebraic function field at a place P, and give the definition of zero and pole, justifying the name of "function" field. We finally give the definition of *divisors* and recall the usual results of the theory: the link between the degree and the dimension of a divisor, the definition of the *genus* of an algebraic function field, and the Riemann-Roch theorem.

In Section 2.3, we give the model of *algebraic complexity* and explain why it is suitable for our work. We also give the usual asymptotic notations *big O* and *little o*, that are used in the thesis to express asymptotic results.

Finally in Section 2.4, we give references and recall the complexity of some classic and very important routines: the Brent-Kung algorithm for modular composition, minimal polynomial computation, and the Berlekamp-Massey algorithm. These routines are used in some algorithms that are presented in this thesis, especially in Chapters 5 and 7.

Bilinear complexity and Chudnosky^2-type algorithms. In Chapter 3, we present the theory of *bilinear complexity*, an alternative model of complexity used to measure the cost of computing bilinear maps. In this model, only multiplications are counted, which is justified because in practice multiplications are harder to compute than additions. The bilinear complexity of a map is then given by the minimal number of needed multiplications. Karatsuba's algorithm is a practical example of the interest in this complexity model. Indeed, the idea behind this algorithm is to multiply two degree 1 polynomials

$$A = a_1X + a_0 \text{ and } B = b_1X + b_0$$

with only three products

$$c_0 = a_0b_0,$$
$$c_1 = (a_0 + a_1)(b_0 + b_1),$$

and

$$c_\infty = a_1b_1,$$

instead of the four classic ones a_0b_0, a_0b_1, a_1b_0 and a_1b_1 as follows:

$$AB = c_\infty X^2 + (c_1 - c_\infty - c_0)X + c_0.$$

In Section 3.1, after rigorously defining the bilinear complexity for a bilinear map Φ using *bilinear formulas*, we explain how it is equivalent to the rank of the tensor corresponding to Φ. We also present the *symmetric* version of the bilinear complexity, which counts the minimal number of symmetric multiplications that we need in order to compute Φ, and that is also studied in the litterature. We are in particular interested in the multiplication in finite field extensions and in understanding the (symmetric) bilinear complexity of this bilinear map.

In Section 3.2, we recall the principle of evaluation-interpolation and present an algorithm due to Chudnovsky and Chudnovsky [CC88] based on evaluation and interpolation on the places of

an algebraic function field. This seminal algorithm gives an asymptotic *linear* bound on both the bilinear complexity and the symmetric bilinear complexity of the product in finite field extensions. It was extensively studied, giving birth to many improvements [BR04, CÖ10, Ran12] that we succinctly present.

However, Chudnovsky and Chudnovsky's algorithm only gives good asymptotic bilinear formulas, hence we also give in Section 3.3 an algorithm due to Barbulescu, Detrey, Estibal and Zimmermann [BDEZ12] to compute the bilinear complexity in small dimension. Their algorithm enumerates all bilinear formulas of a given length, and can thus be used to find the minimal length of a formula for some bilinear map Φ, *i.e.* the bilinear complexity of Φ. The algorithm is explained in details, and some improvements due to Covanov [Cov19] are mentioned.

Hypersymmetric bilinear complexity. In Chapter 4, we investigate new types of interesting complexity models, and provide results both in small dimension and asymptotically. In Section 4.1, we generalize the notion of bilinear complexity to the product of $s \geq 2$ variables

$$x_1 \times x_2 \times \cdots \times x_{s-1} \times x_s$$

in a finite field extension $\mathbb{F}_{p^k}$. Adapting the algorithm of Chudnovsky and Chudnovsky to this case, we show in Section 4.3 that this so called *multilinear complexity* is still linear in the degree k of the extension [RR21], as is the case with the classic bilinear complexity.

In this chapter, we define a new kind of complexity called the *hypersymmetric bilinear complexity*, that is inspired by the usual symmetric bilinear complexity, where an additional symmetry property is asked. We provide an *ad hoc* algorithm, inspired by the algorithm of Barbulescu, Detrey, Estibal and Zimmermann, to compute this complexity in small dimension. We explain our algorithm in details and provide experimental results concerning the hypersymmetric bilinear complexity of the multiplication in finite field extensions in Section 4.2. Using *universal formulas*, *i.e.* formulas that are true for almost any prime p, we also give theoretical results on the hypersymmetric complexity in finite field extensions

$$\mathbb{F}_{p^k}$$

and truncated polynomials algebras

$$\mathbb{F}_p[T]/(T^k)$$

in small dimension k, that generalize known results for the bilinear complexity. Aditionally, we obtain the asymptotic linearity in k of the hypersymmetric complexity of the multiplication in a finite field extension $\mathbb{F}_{p^k}$ as a corollary of the same result for multilinear complexity.

1.2.2 Efficient arithmetic in a lattice of finite fields

In Part II, we study how to deal with multiple finite fields at once, in what we call a *lattice of compatibly embedded finite fields*. This is the equivalent to asking how to compute in the algebraic closure

$$\bar{\mathbb{F}}_p = \bigcup_{k \geq 1} \mathbb{F}_{p^k}$$

of the base field $\mathbb{F}_p$.

Isomorphism algorithms. Chapter 5 is dedicated to the isomorphism problem, which asks how to efficiently compute an isomorphism (or more generally an embedding)

$$K \hookrightarrow L$$

between two finite fields K and L. In Section 5.1, we present the isomorphism problem and, following [BDFD+17], we divide it in two parts, the *embedding description problem* and the *embedding evaluation problem*. The embedding description problem consists in finding elements $\alpha \in K$ and $\beta \in L$ such that

$$K = \mathbb{F}_p(\alpha)$$

and such that there exists an embedding $\phi : K \hookrightarrow L$ mapping α to β. Then, knowing α and β, the embedding evaluation problem consists in efficiently evaluating ϕ. We first deal with the description problem and present the naive algorithm, based on polynomial factorization, in Section 5.1.2. This algorithm plays an important role in Chapter 6.

In Section 5.2, we present a more elaborate algorithm due to Allombert [All02a] and inspired by Lenstra's work [LJ91], that we call the Lenstra-Allombert algorithm. The Lenstra-Allombert algorithm is based on Kummer theory, the study of certain field extensions, and works with primitive roots of unity. If a primitive n-th root of unity ζ_n is in $\mathbb{F}_{p^n}$, then the description of the algorithm is simpler and is given in Section 5.2.1. Otherwise, we need to artificially add ζ_n to $\mathbb{F}_{p^n}$, and this leads to the study of the algebras

$$A_n = \mathbb{F}_{p^n} \otimes \mathbb{F}_p(\zeta_n),$$

that we call *Kummer algebras*. The elements α and β describing the embedding given by the Lenstra-Allombert algorithm are deduced from the solutions of equations of the form

$$(\sigma \otimes 1)(x) = (1 \otimes \zeta)x$$

in Kummer algebras. We call these equations Hilbert 90. Kummer algebras, together with the solutions of Hilbert 90, are extensively studied in Section 5.2.2, because they are also of central importance in Chapter 7.

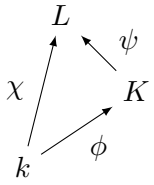
Using the results of Section 5.2.2, we explain in Section 5.2.3 how to derive the elements α and β from the solutions of Hilbert 90. Finally, we present the known techniques to compute solutions of Hilbert 90 in Section 5.2.4. The Lenstra-Allombert algorithm serves as the building block of the algorithms in Chapter 7. Finally, the standard techniques that are used to answer the embedding evaluation problem are presented in Section 5.3.

From a single finite field to plenty: lattices of embeddings. In Chapter 6, we investigate *the compatibility problem*, that asks how to compute embeddings between potentially many more that two finite fields, in a compatible way, *i.e.* so that the diagrams made of the embeddings always commute. This can be expressed in other words: given any three finite field extensions

$$k \subset K \subset L$$

and embeddings $\phi : k \to K$, $\psi : K \to L$, $\chi : k \to L$ between them, we want the equality

$$\chi = \psi \circ \phi.$$



The general problem, as well as additional sub-goals, are presented in Section 6.1.

In Section 6.2, we present a first solution, based on the Conway polynomials [Par, Sch92]. These polynomials are used to define the finite field extensions and possess the very interesting property of being *norm-compatible*, meaning that taking the norm of the root of a Conway polynomial sends it to the root of another Conway polynomial. This provides easy-to-compute embeddings: indeed the embedding description problem is solved by computing a norm. Naturally, before using Conway polynomials, we first need to compute them. This is a problem because we do not know any efficient algorithm to compute Conway polynomials. As a consequence, Conway polynomials are usually precomputed up to some degree d in most computer algebra systems, and it is no longer possible, or computationally expensive, to compute embeddings with finite field extensions that have a degree larger than this given degree d.

In Section 6.3, we introduce another solution to the compatibility problem, called the Bosma-Canon-Steel framework [BCS97]. This framework allows us to use any given polynomial to define our finite field extensions, and is *incremental*, meaning that we can always add more extensions to our data structure, without having to recompute anything. It is much more flexible than Conway polynomials, and is thus a very interesting alternative to them. We describe it in details in Section 6.3.1. This framework was first available in the computer algebra system Magma [BCP97] since at least 1997. To the best of my knowledge, the only additional computer algebra system using the Bosma-Canon-Steel framework is Nemo [H+16]. The framework was implemented in Fall 2019, when I was kindly invited to the university of Kaiserslautern by the developers of Nemo. All the implementation details, as well as experimental results, are given in Section 6.3.2.

Standard lattice of compatibly embedded finite field. Finally, in Chapter 7, we construct a new method [DFRR19] for computing lattices of compatibly embedded finite fields, that is halfway between the Conway polynomials and the Bosma-Canon-Steel framework. The central element of this construction is the Lenstra-Allombert embedding algorithm. In Section 7.1, we explain how this algorithm can be used to produce compatible embeddings, by choosing compatible solutions of (H90), and the new challenges it raises.

In Section 7.2, we show that some large Kummer algebras called *complete Kummer algebras*, that can be described by

$$A_{p^a-1} = \mathbb{F}_{p^{p^a-1}} \otimes \mathbb{F}_p(\zeta_{p^a-1}) = \mathbb{F}_{p^{p^a-1}} \otimes \mathbb{F}_{p^a},$$

admit special solutions of (H90), that we call *standard solutions*. We then prove that from these standard solutions in the complete algebra A_{p^a-1}, we can deduce solutions of (H90) in every Kummer algebra

$$A_n = \mathbb{F}_{p^n} \otimes \mathbb{F}_p(\zeta_n)$$

such that $[\mathbb{F}_p(\zeta_n) : \mathbb{F}_p] = a$. We see that the solutions derived from the standard solution also share some special properties, and that all these solutions can be used to obtain compatible embeddings between all the associated finite field extension $\mathbb{F}_{p^n}$.

In Section 7.2.2, we show that if $a \mid b$, the solutions of (H90) in the complete Kummer algebras A_{p^a-1} and A_{p^b-1} can also be linked by some operator that we call *scalar norm operator* and

that essentially acts like the usual norm of finite fields. We use these results in Section 7.3 to obtain compatible embeddings between arbitrary finite field extensions $\mathbb{F}_{p^m}$ and $\mathbb{F}_{p^n}$, provided that $m \mid n$.

We analyze the complexity of this new framework in Section 7.4 and provide an implementation in the Julia programming language [Jul], using Nemo. We see that our construction is practical, and that embeddings are computed in a reasonable time.

Part I

Efficient arithmetic in a single finite field

Chapter 2

Preliminaries

Throughout all this document, we will use a lot of results from algebra. This chapter is here to sum up these results and try to maintain the illusion that this thesis is self-contained. Our references for standard results in algebra are [Lan04] or [Per96]. The reader familiar with the notions of finite fields, algebraic function fields, or complexity model may very well skip this chapter.

Contents

2.1 Finite fields 28
2.1.1 Finite field structure 28
2.1.2 Subfields and field extensions 28
2.2 Algebraic function fields 30
2.2.1 Places 31
2.2.2 Independence of valuations 34
2.2.3 Divisors 34
2.3 Complexity models 37
2.3.1 Algebraic complexity 38
2.3.2 Landau notations 38
2.4 Fundamental algorithms 39
2.4.1 Finite field arithmetic 39
2.4.2 Classic routines 40

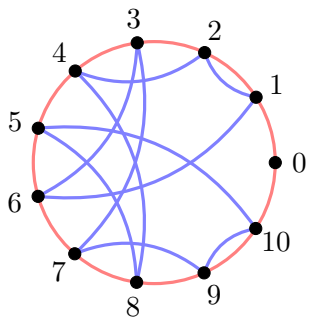


Figure 2.1: Cyclic group structure of $(\mathbb{F}_{11}, +)$ (red) and $(\mathbb{F}_{11}^{\times}, \times)$ (blue).

2.1 Finite fields

Finite fields are ubiquitous in cryptography and coding theory, probably because their field structure, a rigid one, allows to understand how they work, and their finiteness makes them easier to represent on a computer. They are also everywhere in this thesis, and are probably on almost every paper I wrote on during these last three years. They are quite important. A detailed book about finite fields is [LN97].

2.1.1 Finite field structure

A *finite field* is a field $\mathbf{k}$ whose cardinality is finite. The first examples of finite fields are the rings

$$\mathbb{Z}/p\mathbb{Z}$$

with $p \in \mathbb{N}$ a prime number. More generally, we denote by $\mathbb{F}_q$ the finite field with q elements. The cardinality of a finite field is very well understood.

Proposition 2.1.1. *There exists a unique (up to isomorphism) finite field of cardinality $q = p^l$ for each prime number $p \in \mathbb{N}$ and integer $l \geq 1$, and every finite field has cardinality of the form $q = p^l$.*

Let $q = p^l$ a prime power, there are several ways of representing *the* finite field with q elements, but the one that we will almost always have in mind is the following.

Proposition 2.1.2. *Let $P \in \mathbb{F}_p[x]$ be an irreducible polynomial of degree l with coefficients in $\mathbb{F}_p$. Then*

$$\mathbb{F}_p[x]/(P(x))$$

is a finite field with $q = p^l$ elements.

We often write

$$\mathbb{F}_q \cong \mathbb{F}_p[x]/(P(x)) \cong \mathbb{F}_p(\alpha)$$

in order to say that we work with a finite field of q elements, represented by the quotient $\mathbb{F}_p[x]/(P(X))$, and where the projection of x in the quotient is denoted by $\alpha = \bar{x}$.

2.1.2 Subfields and field extensions

The finite field $\mathbb{F}_{p^l}$ is a field extension of $\mathbb{F}_p$ of dimension l, *i.e.* it is a $\mathbb{F}_p$-vector space of dimension l. When dealing with the vector space structure of $\mathbb{F}_{p^l} = \mathbb{F}_p(\alpha)$, we almost always choose to work with the canonical basis

$$1, \alpha, \alpha^2, \ldots, \alpha^{l-1}.$$

Other interesting types of bases exist, such as for example normal bases [Gao93], but we always specify the basis when it is not clear from the context. Given $q = p^m$ a prime power and $l \in \mathbb{N}$ an integer, we also write

$$\mathbb{F}_{q^l}$$

the field with $q^l = p^{ml}$ elements. We have

$$\mathbb{F}_{q^l} \cong \mathbb{F}_{p^{lm}},$$

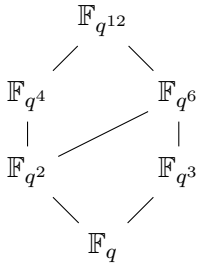


Figure 2.2: The subfields of $\mathbb{F}_{q^{12}}$. Two fields are linked if one is a subfield of the other.

but the difference is that we see $\mathbb{F}_{q^l}$ as an extension of $\mathbb{F}_q$ of dimension l, and not as an extension of the prime field $\mathbb{F}_p$. Again, we usually think that our field $\mathbb{F}_{q^l}$ is represented as

$$\mathbb{F}_{q^l} = \mathbb{F}_q[x]/(P(x)),$$

where $P(x) \in \mathbb{F}_q[x]$ is an irreducible polynomial of degree l with coefficients in the base field $\mathbb{F}_q$. The subfields of $\mathbb{F}_{q^l}$ are also well understood.

Proposition 2.1.3. *Let $q = p^m$ be a prime power and $l \in \mathbb{N}$ an integer. Then there is an extension $\mathbb{F}_{q^m}$ of $\mathbb{F}_q$ of degree m included in $\mathbb{F}_{q^l}$*

$$\mathbb{F}_{q^m} \subset \mathbb{F}_{q^l}$$

if and only if m divides l. The elements in this subfield of $\mathbb{F}_{q^l}$ are the roots of the polynomial

$$x^{q^m} - x$$

in $\mathbb{F}_{q^l}$.

Figure 2.2 describes the subfields of $\mathbb{F}_{q^{12}}$, as an illustration of Proposition 2.1.3. The $\mathbb{F}_q$-automorphisms of the extension $\mathbb{F}_{q^l}$ are given by the following result.

Proposition 2.1.4. *Let $q = p^m$ be a prime power and $l \in \mathbb{N}$ an integer. The group of $\mathbb{F}_q$-automorphisms of $\mathbb{F}_{q^l}$ is a cyclic group of order l generated by*

$$\sigma : t \mapsto t^q.$$

Let $u \in \mathbb{F}_{q^l}$, the conjugates of u are the elements

$$\sigma(u), \sigma^2(u), \ldots, \sigma^{l-1}(u)$$

and the orbit of u is the set

$$\left\{u, \sigma(u), \ldots, \sigma^{l-1}\right\}.$$

This orbit might have any length m dividing l. The orbit of u is of length exactly m if and only if the smallest subfield u belongs to is the subfield $\mathbb{F}_{q^m}$ of $\mathbb{F}_{q^l}$. We also sometimes write

$$u^\sigma = \sigma(u).$$

Two maps, defined with the conjugates of a given element, will play a very important role, the *trace* and the *norm*.

Definition 2.1.5 (Trace and norm)**.** Let q a prime power and

$$\mathbb{F}_{q^l}/\mathbb{F}_q$$

an extension of degree l, let G be the group of $\mathbb{F}_q$-automorphisms of $\mathbb{F}_{q^l}$, and let $u \in \mathbb{F}_{q^l}$. Then the *trace* of u is

$$\mathrm{Tr}_{\mathbb{F}_{q^l}/\mathbb{F}_q}(u) = \sum_{\sigma \in G} u^\sigma$$

and the *norm* of u is

$$N_{\mathbb{F}_{q^l}/\mathbb{F}_q}(u) = \prod_{\sigma \in G} u^\sigma.$$

We may only write Tr or N when the extension considered is clear from the context. The trace over the field $\mathbb{F}_q$ is a $\mathbb{F}_q$-linear map, and the norm is a multiplicative map, they are also both *transitive*, as described in the next proposition.

Proposition 2.1.6. *Let $q \in \mathbb{N}$ a prime power and $a \,|\, b \,|\, c$ three integers, giving the tower of extensions that follows.*

$$\begin{array}{c} \mathbb{F}_{q^c} \\ | \\ \mathbb{F}_{q^b} \\ | \\ \mathbb{F}_{q^a} \end{array}$$

Let $u \in \mathbb{F}_{q^c}$, then

$$\mathrm{Tr}_{\mathbb{F}_{q^c}/\mathbb{F}_{q^a}}(u) = \mathrm{Tr}_{\mathbb{F}_{q^b}/\mathbb{F}_{q^a}}(\mathrm{Tr}_{\mathbb{F}_{q^c}/\mathbb{F}_{q^b}}(u))$$

and

$$N_{\mathbb{F}_{q^c}/\mathbb{F}_{q^a}}(u) = N_{\mathbb{F}_{q^b}/\mathbb{F}_{q^a}}(N_{\mathbb{F}_{q^c}/\mathbb{F}_{q^b}}(u)).$$

Finally, the map

$$(x, y) \mapsto \mathrm{Tr}(xy)$$

is a non-degenerate bilinear map that we will also sometimes write as

$$\langle x, y \rangle = \mathrm{Tr}(xy).$$

2.2 Algebraic function fields

Together with algebraic curves, algebraic function fields are a way of describing the algorithms of Chapters 3 and 4. In this document, we choose to use the algebraic function field point of view. The function fields we need are constructed on top of finite fields, so we present the theory with that context in mind. For this section, our reference is [Sti09]. In all the section, $\mathbf{k}$ is a finite field of characteristic p.

2.2.1 Places

Let us first define what an algebraic function field is, together with important notions leading to the definition of *places*.

Definition 2.2.1 (Algebraic function field)**.** An *algebraic function field* F of one variable over $\mathbf{k}$ is an extension field

$$F/\mathbf{k}$$

such that F is a finite algebraic extension of $\mathbf{k}(x)$ for some element $x \in F$ which is transcendental over $\mathbf{k}$.

From now on, the notation F will represent an algebraic function field over $\mathbf{k}$. Since it is not critical to the theory, we also assume that $\mathbf{k}$ is algebraically closed in $\mathbf{k}$ for simplicity.

Definition 2.2.2 (Valuation ring)**.** A *valuation ring* $\mathcal{O}$ of the function field $F/\mathbf{k}$ is a ring

$$\mathcal{O} \subset F$$

with the following properties

1. $\mathbf{k} \subsetneq \mathcal{O} \subsetneq F$;
2. for all $z \in F$, we have $z \in \mathcal{O}$ or $z^{-1} \in \mathcal{O}$.

Proposition 2.2.3. *Let $\mathcal{O}$ be the a valuation ring of the function field F.*

(a) The ring $\mathcal{O}$ is local, i.e. *it has a unique maximal ideal that is given by*

$$P = \mathcal{O} \setminus \mathcal{O}^\times$$

where $\mathcal{O}^\times$ is the group of units of the ring $\mathcal{O}$.

(b) For any nonzero $x \in F$, we have

$$x \in P \iff x^{-1} \notin \mathcal{O}.$$

Theorem 2.2.4. *Let $\mathcal{O}$ be the valuation ring of the function field F and P be its unique maximal ideal. Then*

(a) The ideal P is principal.

(b) If $P = t\mathcal{O}$ then any nonzero $z \in F$ has a unique representation of the form

$$z = t^n u$$

with $n \in \mathbb{Z}$ and $u \in \mathcal{O}^\times$

(c) The ring $\mathcal{O}$ is a principal ideal domain. More precisely, if $P = t\mathcal{O}$ and

$$\{0\} \neq I \subseteq \mathcal{O}$$

is an ideal then

$$I = t^n\mathcal{O}$$

for some $n \in \mathbb{N}$.

A ring having the properties described in Theorem 2.2.4 is called a *discrete valuation ring*.

Definition 2.2.5 (Place)**.** A *place* P of F is the maximal ideal of some valuation ring $\mathcal{O}$ of F.

Definition 2.2.6 (Prime element)**.** Let $\mathcal{O}$ be a valuation ring and P its maximal ideal. Any element $t \in P$ such that

$$P = t\mathcal{O}$$

is called a *prime element* for P. It is also sometimes called a *local parameter* or a *uniformizing variable*.

If $\mathcal{O}$ is a valuation ring of F and if P is its maximal ideal, then $\mathcal{O}$ is uniquely determined by P, indeed we have

$$\mathcal{O} = \left\{ z \in F \mid z^{-1} \notin P \right\}$$

thanks to Proposition 2.2.3(b). Thus, the notions of valuation rings and places are essentially equivalent. There is a third way of describing a place, given by discrete valuations.

Definition 2.2.7 (Discrete valuation)**.** A *discrete valuation* of F is a function

$$v : F \to \mathbb{Z} \cup \{\infty\}$$

with the following properties:

1. $v(x) = \infty \Leftrightarrow x = 0$;
2. for any $x, y \in F$, $v(xy) = v(x) + v(y)$;
3. for any $x, y \in F$, $v(x + y) \geq \min(v(x), v(y))$;
4. there exists an element $z \in F$ with $v(z) = 1$;
5. for any nonzero element $a \in \mathbf{k}$, $v(a) = 0$.

Here, the symbol ∞ means an element that is not in $\mathbb{Z}$, such that $\infty + \infty = \infty + n = \infty$ and $\infty > m$ for any $m, n \in \mathbb{Z}$.

Definition 2.2.8. To any place $P \in \mathbb{P}_F$, we associate a function

$$v_P : F \to \mathbb{Z} \cup \{\infty\}$$

that is in fact a discrete valuation ring. Let t be a prime element for P. Then every nonzero element $z \in F$ has a unique representation

$$z = t^n u$$

with $u \in \mathcal{O}^\times$ and $n \in \mathbb{Z}$. We define

$$v_P(z) \stackrel{\text{def}}{=} n$$

and

$$v_P(0) \stackrel{\text{def}}{=} \infty.$$

This definition does not depend on the prime element t that was chosen. It allows us to give the third equivalent way of describing a place of F.

Theorem 2.2.9. *Let F be a function field.*

(a) For any place $P \in \mathbb{P}_F$, the function v_P defined above is a discrete valuation of F. Moreover, we have

$$\begin{aligned} \mathcal{O}_P &= \{z \in F \mid v_P(z) \geq 0\}, \\ \mathcal{O}_P^\times &= \{z \in F \mid v_P(z) > 0\}, \\ P &= \{z \in F \mid v_P(z) = 0\}. \end{aligned}$$

An element $x \in F$ is a prime element for P if and only if $v_P(x) = 1$.

(b) Conversely, if v is a discrete valuation of F, then the set

$$P = \{z \in F \mid v(z) > 0\}$$

is a place of F, and

$$\mathcal{O}_P = \{z \in F \mid v(z) \geq 0\}$$

is the corresponding valuation ring.

We let $\mathbb{P}_F$ be the set of places of F. If $P \in \mathbb{P}_F$ is a place of F, we denote by $\mathcal{O}_P$ the corresponding valuation ring. Since P is a maximal ideal of $\mathcal{O}$, we also know that the quotient ring

$$F_P = \mathcal{O}_P / P$$

is a field. We call this field F_P the *residue class field* of P.

Definition 2.2.10 (Residue class map)**.** Let $P \in \mathbb{P}_F$ be a place of F. For $x \in F$, we let

$$x(P)$$

be the residue class of x modulo P. If $x \notin \mathcal{O}_P$, we define $x(P) = \infty$. The map from F to $F_P \cup \{\infty\}$

$$x \mapsto x(P)$$

is called the *residue class map.*

Definition 2.2.11 (Degree of a place)**.** Let $P \in \mathbb{P}_F$ be a place of F. The residue class field F_P is a finite extension of $\mathbf{k}$ and we call

$$\deg P = [F_P : \mathbf{k}]$$

the *degree* of P.

In fact, the degree of a place if always finite. If P is a place of degree 1, then its residue class field F_P is equal to

$$F_P = \mathbf{k}.$$

The residue class map then maps F to $\mathbf{k} \cup \{\infty\}$. In particular, if $\mathbf{k}$ is algebraically closed, any place has degree 1, thus we can interpret an element $z \in F$ as a function

$$\begin{array}{rccc} z: & \mathbb{P}_F & \to & \mathbf{k} \cup \{\infty\} \\ & P & \mapsto & z(P) \end{array}.$$

This justifies the name *function field* for F. With this interpretation, places of degree 1 are viewed as points and elements in $\mathbf{k}$ are viewed as constant functions. This is also a reason behing the following terminology.

Definition 2.2.12 (Zero and pole)**.** Let $z \in F$ be an element in the function field F and $P \in \mathbb{P}_F$ a place of F. We say that P is a *zero* of z if $v_P(z) > 0$ and that P is a *pole* of z if $v_P(z) < 0$. If $v_P(z) = n > 0$, then P is a *zero of order* n; if $v_P(z) = -n < 0$, then P is a *pole of order* n.

Note that if P is a zero of $z \in F$, we have $z(P) = 0$, while we have $z(P) = \infty$ if P is a pole of z. Another important result states that every element not in $\mathbf{k}$ yields a non-constant function.

Proposition 2.2.13. *Let F be a function field and $z \in F$ an element in F that is not in $\mathbf{k}$. Then z has at least one zero and one pole. In particular, this proves that the set $\mathbb{P}_F$ of places is not empty.*

2.2.2 Independence of valuations

Before talking about the central notions of divisors and Riemann-Roch spaces, we must mention one important theorem called the *weak approximation theorem* (also refered to as *theorem of independence*). It essentially says that if we have pairwise distinct discrete valuations $v_1, \dots, v_n$ of F and an element $z \in F$ for which we know the values $v_1(z), \dots, v_{n-1}(z)$, then we cannot conclude anything about the value $v_n(z)$.

Theorem 2.2.14. *Let F be a function field, $P_1, \dots, P_n \in \mathbb{P}_F$ be pairwise distinct places of F, $x_1, \dots, x_n \in F$ be elements in F and $r_1, \dots, r_n \in \mathbb{Z}$ be integers. Then there is some $x \in F$ such that*

$$v_{P_i}(x - x_i) = r_i \text{ for all } 1 \leq i \leq n.$$

The name "approximation theorem" comes from the fact that $v_{P_i}(x - x_i)$ can be interpreted as some "distance" between x and x_i. With this interpretation, Theorem 2.2.14 says that the distances between x and x_i can be made simultaneously arbitrarily small using *one* element x. This theorem allows us to mention important results.

Corollary 2.2.15. *Any function field F has infinitely many places.*

Proposition 2.2.16. *Let F be a function field and $P_1, \dots, P_r$ be the zeros of some element $x \in F$. Then*

$$\sum_{i=1}^{r} v_{P_i}(x) \cdot \deg P_i \leq [F : \mathbf{k}(x)] \,.$$

Corollary 2.2.17. *In a function field F, any nonzero element $x \in F$ has only finitely many zeros and poles.*

2.2.3 Divisors

Now that we know what places are, we can define divisors. In this section, we keep the notation of the last section: we let F be an algebraic function field over a finite field $\mathbf{k}$ of characteristic p.

Definition 2.2.18 (Divisor)**.** The *divisor group* $\mathrm{Div}(F)$ of F is defined as the free abelian group generated with the places of F. The elements of $\mathrm{Div}(F)$ are called the *divisors* of F. In other words, a divisor is a formal sum

$$D = \sum_{P \in \mathbb{P}_F} n_P \cdot P$$

with $n_P \in \mathbb{Z}$ and $n_P = 0$ for all but finitely many places P.

Definition 2.2.19 (Support)**.** Let

$$D = \sum_{P \in \mathbb{P}_F} n_P \cdot P$$

be a divisor of F. We define the *support* of D as

$$\operatorname{supp} D = \{P \in \mathbb{P}_F \,|\, n_P \neq 0\}.$$

Two divisors

$$D = \sum_{P \in \mathbb{P}_F} n_P \cdot P$$

and

$$D' = \sum_{P \in \mathbb{P}_F} n'_P \cdot P$$

in $\operatorname{Div}(F)$ are added coefficient-wise:

$$D + D' = \sum_{P \in \mathbb{P}_F} (n_P + n'_P) \cdot P$$

and the zero element is the divisor

$$0 \stackrel{\text{def}}{=} \sum_{P \in \mathbb{P}_F} n_P \cdot P$$

with $n_P = 0$ for all places $P \in \mathbb{P}_F$. If

$$D = \sum_{P \in \mathbb{P}_F} n_P \cdot P$$

is a divisor $\operatorname{Div}(F)$ and $Q \in \mathbb{P}_F$ is a place of F, we let

$$v_Q(D) = n_Q$$

be the coefficient multiplying Q in the formal sum D. We then have

$$\operatorname{supp} D = \{P \in \mathbb{P}_F \mid v_P(D) \neq 0\}.$$

We then define a partial ordering on $\operatorname{Div}(F)$ by

$$D_1 \leq D_2 \stackrel{\text{def}}{\iff} v_P(D_1) \leq v_P(D_2) \text{ for all } P \in \mathbb{P}_F.$$

A divisor $D \geq 0$ is called *positive* (or *effective*).

Definition 2.2.20 (Degree)**.** Let $D \in \operatorname{Div}(F)$ be a divisor, its *degree* is defined by

$$\deg(D) = \sum_{P \in \mathbb{P}_F} v_P(D) \deg(P)$$

and yields a homomorphism $\deg : \operatorname{Div}(F) \to \mathbb{Z}$.

Definition 2.2.21. Let $x \in F$ be a nonzero element in F, $Z \subset \mathbb{P}_F$ be the set of its zeros and $N \subset \mathbb{P}_F$ be the set of its poles. Then, we define

$$\begin{aligned}(x)_0 &= \sum_{P \in Z} v_P(x)P, \text{ the } \textit{zero divisor} \text{ of } x,\\ (x)_\infty &= \sum_{P \in N} (-v_P(x))P, \text{ the } \textit{pole divisor} \text{ of } x,\\ (x) &= (x)_0 - (x)_\infty, \text{ the } \textit{principal divisor} \text{ of } x.\end{aligned}$$

We have $(x)_0 \geq 0, (x)_\infty \geq 0$ and

$$(x) = \sum_{P \in \mathbb{P}_F} v_P(x)P.$$

If $x \in K$, then x does not have zeros nor poles and the sum is then empty, conversely a function in $F \setminus \mathbf{k}$ has at least one zero and one pole, we thus have

$$x \in \mathbf{k} \Leftrightarrow (x) = 0.$$

Definition 2.2.22 (Equivalence)**.** Let $D, D' \in \mathrm{Div}(F)$ be two divisors of F. We can define an equivalence relation by

$$D \sim D' \iff \text{ there exists } x \in F \text{ such that } D = D' + (x).$$

In that case, we say that D and D' are *linearly equivalent*, or just *equivalent*.

Principal divisors play a crucial role in the algebraic function field theory. They allow us to define spaces that play a fundamental role.

Definition 2.2.23. Let $D \in \mathrm{Div}(F)$ be a divisor of F. We let

$$L(D) = \{x \in F \mid (x) \geq -D\} \cup \{0\}.$$

This definition can be interpreted in the following way: if

$$D = \sum_{i=1}^{r} n_i P_i - \sum_{j=1}^{s} m_j Q_j,$$

with $n_i > 0$ and $m_j > 0$, then $L(D)$ consists of all elements $x \in F$ such that

1. for all $1 \leq j \leq s$, x has a zero of order at least m_j at Q_j;
2. the places $P_1, \ldots, P_r$ are the only poles of x, and for all $1 \leq i \leq r$, the order of the pole P_i is bounded by n_i.

We recall some properties of the spaces $L(D)$ in Proposition 2.2.24.

Proposition 2.2.24. *Let $D \in \mathrm{Div}(F)$ be a divisor.*

(a) $L(D)$ is a finite-dimensionnal $\mathbf{k}$-vector space.

(b) $L(D) \neq \{0\}$ if and only if there exists a divisor $D' \sim D$ with $D' \geq 0$.

(c) If $D' \sim D$, then $L(D)$ and $L(D')$ are isomorphic as $\mathbf{k}$-vector spaces.

(d) $x \in L(D)$ if and only if $v_P(x) \geq -v_P(D)$ for all $P \in \mathbb{P}_F$.

(e) $L(0) = \mathbf{k}$.

(f) If $\deg(D) < 0$ then $L(D) = \{0\}$. In particular if $D < 0$ then $L(D) = \{0\}$.

Definition 2.2.25 (Dimension)**.** Let $D \in \mathrm{Div}(F)$ be a divisor of F. The *dimension* of D is denoted by $\ell(D)$ and is defined by

$$\ell(D) = \dim L(D).$$

We can now define the most important invariant of a function field F, its *genus*.

Definition 2.2.26 (Genus)**.** Let F be a function field. The *genus* g of F is the nonnegative integer defined by

$$g = \max\{\deg D - \ell(D) + 1 \mid D \in \mathrm{Div}(F)\}.$$

Definition 2.2.27 (Index of specialty)**.** Let $D \in \mathrm{Div}(F)$ be a divisor of F, the nonnegative integer

$$i(D) = \ell(D) - \deg(D) + g - 1$$

is called the *index of specialty* of D. A divisor D such that $i(D) = 0$ is called *non-special*; a divisor D such that $i(D) > 0$ is called *special*.

Definition 2.2.28 (Canonical divisor)**.** Let F be a function field of genus g. A divisor W of degree $\deg W = 2g - 2$ and dimension $\ell(W) = g$ is called a *canonical divisor*.

In fact, all canonical divisors are equivalent, and they are involved in the very important Riemman-Roch theorem.

Theorem 2.2.29 (Riemann-Roch)**.** *Let $W \in \mathrm{Div}(F)$ be a canonical divisor of F. Then, for any divisor $D \in \mathrm{Div}(F)$, we have*

$$\ell(D) = \deg(D) + 1 - g + \ell(W - D).$$

Remark 2.2.30. Note that with Definition 2.2.27, Theorem 2.2.29 means that

$$i(D) = \ell(W - D).$$

2.3 Complexity models

When studying algorithms, it is of central interest to understand how our algorithms *scale*, *i.e.* to understand how they perform if the size of the input is getting larger and larger. Complexity theory studies this phenomenon and gives us models of computation in order to quantify the behaviour of our algorithms. Depending on the situation, not all models are relevant, and one has to balance between the concreteness of a model and its ease of use. An extreme viewpoint is to specify an operating system with a compiled programming language and to compare the running time or the memory requirements between algorithms. The advantage of such a model is that it is very concrete, but it is also its main disadvantage because it makes the model hard to use. Thus, there exist other models of *idealized* computers, such as Turing machines [Pap03], random access machines, or the algebraic complexity model. We use the latter, that we present in more details in the next section.

2.3.1 Algebraic complexity

This model is widely presented in [BCS13], we only give a brief presentation of the subject. This model assumes that we use an abstract computer that is able to perform operations in some base field $\mathbf{k}$ at a constant, unit cost. We also assume that accessing the memory of the computer is free. Algebraic complexity is especially useful with algorithms dealing with algebraic structures. This is very handy for us, since we usually work with algebras

$$(\mathcal{A}, +, \times, \cdot)$$

over some base field $\mathbf{k}$. As an example, with this model, the complexity of an addition in the extension field

$$\mathbb{F}_4 = \mathbb{F}_2[T]/(T^2 + T + 1) = \mathbb{F}_2(x)$$

where $x = \bar{T}$, if elements are represented in the basis $\{1, x\}$, is 2, because we only need 2 additions in $\mathbb{F}_2$ to perform an addition in $\mathbb{F}_4$. Indeed, if

$$a = a_0 + a_1 x \in \mathbb{F}_4$$

and

$$b = b_0 + b_1 x \in \mathbb{F}_4,$$

we have

$$a + b = (a_0 + b_0) + (a_1 + b_1)x.$$

In the case of a multiplication, we have

$$ab = (a_0 b_0 + a_1 b_1) + (a_0 b_1 + a_1 b_0)x,$$

so the complexity of a multiplication in $\mathbb{F}_4$ (at least with this formula) is 6, because we need 4 multiplications and 2 additions in $\mathbb{F}_2$. In the context of finite fields, it makes sense to consider that the cost of an operation is independent of the operands, because the elements have a fixed size; but this is no longer the case in other rings, for example in $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, or $\mathbb{C}$. We could also argue that the different operations in $\mathbf{k}$ should not have the same cost, we thus present an other manner of computing the complexity of an algorithm, that is called *bilinear complexity*, in Chapter 3.

2.3.2 Landau notations

In order to describe the asymptotic behaviour of an algorithm, we use the classical *big O* and *little o* notations O and o. Let $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ be two functions, we write

$$f(x) = O(g(x))$$

if there exist $M \in \mathbb{R}$ and $C > 0$ such that

$$\forall x \geq M, |f(x)| \leq Cg(x).$$

and we write

$$f(x) = o(g(x))$$

if there exist $M \in \mathbb{R}$ and $\varepsilon : \mathbb{R} \to \mathbb{R}$, a function with $\varepsilon(x) \to 0$ when $x \to \infty$, such that

$$\forall x \geq M, |f(x)| \leq \varepsilon(x)g(x).$$

We say that f is equivalent to g and we write

$$f(x) \sim g(x)$$

if

$$f(x) - g(x) = o(g(x))$$

when $x \to \infty$. Finally, we also use the *soft* O notation $\tilde{O}$ to neglect logarithmic factors in the big O notation, we write

$$f(x) = \tilde{O}(g(x))$$

if there exist some k with $f(x) = O(g(x)\log^k(g(x)))$.

2.4 Fundamental algorithms

In this section, we briefly review the fundamental algorithms that are used in the thesis. Unless explicitely stated otherwise, we measure the complexities using the algebraic complexity presented in Section 2.3.1, *i.e.* in number of operations $+, \times, \div$ in some base field $\mathbf{k}$. References for this section are [VZGG13, BCG+17, BDFD+17].

2.4.1 Finite field arithmetic

Since finite fields are so important in this thesis, we first review the complexity of the basic operations in finite fields. We let

$$\mathbf{k} = \mathbb{F}_q$$

be the finite field with q elements, where q is a prime power and we let p be the characteristic of $\mathbf{k}$. We consider $\mathbb{F}_{q^n}$ a finite field extension of degree n of $\mathbf{k}$, and we assume that the elements in $\mathbb{F}_{q^n}$ are represented by univariate polynomials in $\mathbf{k}[x]$. We let $P \in \mathbf{k}[x]$ be the irreducible polynomial defining $\mathbb{F}_{q^n}$, *i.e.* we have

$$\mathbb{F}_{q^n} = \mathbf{k}[x]/(P(x)).$$

Then, the cost of the operations in $\mathbb{F}_{q^n}$ are those of polynomial operations modulo P in $\mathbf{k}[x]$ and depend on polynomial arithmetic. We let $M(n)$ be a function such that polynomials in $\mathbf{k}[x]$ of degree less than n can be multiplied in $M(n)$ operations in $\mathbf{k}$. We assume that the function M has the *superlinearity* property [VZGG13, Chapter 8.3], *i.e.* that for any $m, n \in \mathbb{N} \setminus \{0\}$, we have

$$\begin{cases} M(mn) \geq mM(n) \\ M(n+m) \geq M(m) + M(n) \\ M(n) \geq n \end{cases}.$$

We also assume that $M(mn)$ is in $O(m^{1+\varepsilon}M(n))$ for all $\varepsilon > 0$. Using Fast Fourrier Transform [CT65, SS71], Cantor and Kaltofen [CK91] then proved that we have

$$M(n) \in O(n\log(n)\log\log(n)).$$

Linear algebra operations play an important role too. We denote by ω the *exponent of linear algebra*, *i.e.* a constant such that $n \times n$ matrices in any field $\mathbf{k}$ can be multiplied using $O(n^\omega)$ additions and multiplications in $\mathbf{k}$. One can take

$$\omega < 2.37286$$

using [AW21]; on the other hand, we also suppose that $\omega > 2$. The algorithms achieving the best asymptotic complexity for matrix multiplications are not practical, thus when estimating the complexity of algorithms we sometimes take $\omega = 3$, the complexity coming from the usual formula for matrix multiplication, or $\omega \approx 2.807$ using Strassen's algorithm [Str69].

Adding two elements in $\mathbb{F}_{q^n}$ takes $O(n)$ additions in $\mathbf{k}$. Similarly, adding two polynomials of degree up to s in $\mathbb{F}_{q^n}[T]$ takes $O(s)$ additions in $\mathbb{F}_{q^n}$, thus takes $O(sn)$ additions in $\mathbf{k}$. Multiplying and dividing polynomials of degree at most s in $\mathbb{F}_{q^n}[T]$ is done in $O(M(sn))$ operations in $\mathbf{k}$, using Kronecker's substitution [Moe76, Kal87, VZGG13, VZGS92, Har09]. If $h(T) \in \mathbb{F}_{q^n}[T]$ is a monic polynomial, multiplication in $\mathbb{F}_{q^n}[T]/(h(T))$ is also done in $O(M(sn))$, using the technique in [PS06]. In particular, this means that multiplication in $\mathbb{F}_{q^n}$ costs $O(M(n))$ operations in $\mathbf{k}$. Using the same techniques, the greatest common divisor (gcd) of degree s polynomials in $\mathbb{F}_{q^n}[T]$ and inverses in $\mathbb{F}_{q^n}[T]/(h(T))$ can be computed using $O(M(sn)\log(sn))$ operations in $\mathbf{k}$. Again, this means that inverses in $\mathbb{F}_{q^n}$ can be computed using $O(M(n)\log(n))$ operations in $\mathbf{k}$.

2.4.2 Classic routines

We now give a few standard routines that are used in a lot of algorithms involving finite fields, for example the algorithms presented in Chapters 5 and 7 extensively use these routines. Given polynomials $e, g, h \in \mathbb{F}_{q^n}[T]$ of degree at most s, the modular composition is the problem of computing

$$e(g(T)) \mod h(T).$$

An upper bound on the algebraic complexity is obtained using the Brent-Kung algorithm [BK78]. Following our discussion on the costs of polynomial and matrix multiplication, its cost is $O(s^{(\omega+1)/2}M(n))$ operations in $\mathbf{k}$. In the binary RAM complexity model, the Kedlaya-Umans algorithm [KU11] and its extension [PS13] yield an algorithm with essentially linear complexity in s, n and $\log(q)$. Unfortunately, these algorithms prove to be hard to implement in a competitive way, and Brent and Kung's algorithm seems to outperform them in practice.

By applying transposition techniques [BLS03, DF10, DFS10, BCS13] to Brent and Kung's algorithm, Shoup [Sho94, Sho99] derived an algorithm to compute the minimal polynomial (over $\mathbf{k}$) of an element in $\mathbb{F}_{q^n}$ using $O(n^{(\omega+1)/2})$ operations in $\mathbf{k}$. We say a bit more about these techniques, called *transposition principle* (or *Tellegen's principle*) in Section 5.3.3.

We also sometimes use the Berlekamp-Massey algorithm in order to compute the minimal generating polynomial of a sequence satisfying a linear recurrence relation. The Berlekamp-Massey algorithm takes as input the $2n$ first terms $a_0, \ldots, a_{2n-1} \in \mathbf{k}$ of the sequence for which we know that the minimal generating polynomial has its degree bounded by n. Using rational reconstruction [BCG+17, Chapter 7], the Berlekamp-Massey algorithm outputs the minimal generating polynomial of the sequence $a_0, \ldots, a_{2n-1}$ using $O(M(n)\log(n))$ operations in $\mathbf{k}$.

Chapter 3

Bilinear complexity and Chudnosky^2-type algorithms

We have presented in Section 2.3 an abstract model made to understand the asymptotic behaviour of our algorithms. In this chapter, we present an alternative notion of complexity called *bilinear complexity*, where the focus is on the number of multiplications needed to compute some map.

Contents

3.1 Bilinear complexity 42
3.2 Chudnovsky-Chudnovsky algorithm 46
3.2.1 Evaluation - Interpolation 46
3.2.2 Asymptotic complexity 48
3.3 Algorithmic searches in small dimension 49
3.3.1 Barbulescu, Detrey, Estibals and Zimmerman's algorithm 50

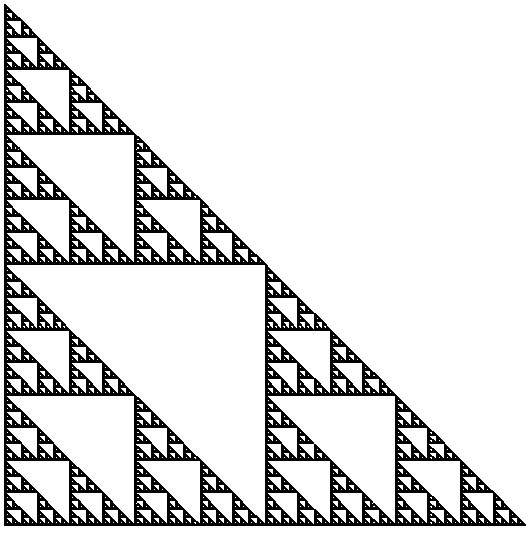


Figure 3.1: Representation of Karatsuba's algorithm complexity.

3.1 Bilinear complexity

In the *algebraic complexity model* [BCS13], we assume that our machine is able to perform any operation in some base field $\mathbf{k}$ in constant, unit time. This is an idealized model made in order to simplify the computation of the complexity of algebraic algorithms. Nevertheless, multiplication of two quantities in $\mathbf{k}$ that are both variable is arguably more expensive than addition, or than multiplication of a variable by a fixed constant. In the context of the computation of bilinear maps, extensive work has been done to reduce the number of 2-variable multiplications involved. Notable examples are Karatsuba's algorithm [Kar63] and Strassen's algorithm [Str69]. Karatsuba's algorithm is based on the fact that the bilinear map associated to the product of two polynomials of degree 1

$$A = a_1X + a_0 \text{ and } B = b_1X + b_0$$

can be computed with three products

$$c_0 = a_0b_0,$$
$$c_1 = (a_0 + a_1)(b_0 + b_1),$$

and

$$c_\infty = a_1b_1,$$

instead of the four classic ones a_0b_0, a_0b_1, a_1b_0 and a_1b_1 as follows:

$$AB = c_\infty X^2 + (c_1 - c_\infty - c_0)X + c_0.$$

It will become clear in Section 3.2.1 why we use the subscript ∞ instead of 2 for $c_\infty = a_1b_1$. Strassen's algorithm exploits a similar idea in the case of 2×2 matrices: only 7 products are used instead of 8 in order to compute a matrix product. Both these algorithms have very practical consequences. Karatsuba's algorithm is used in computer algebra software, when the standard multiplication is no longer optimal, and when the Fast Fourier Transform (FFT) [CT65, SS71] is not yet the fastest. Though Strassen's algorithm does not achieve the best asymptotic complexity (see for example [CW90, AW21]), in practice, when used recursively, it is the fastest strategy available for large symbolic matrix multiplication. Both these questions are treated in [VZGG13]. Thus the idea of minimizing the number of multiplications, even if it means having to compute more additions and substractions, seems a good idea.

The *bilinear complexity* $\mu(\Phi)$ of a bilinear map Φ over $\mathbf{k}$ represents the minimum number of 2-variable multiplications in a formula that computes Φ, discarding the cost of other operations such as addition or multiplication by a constant. In other words, in this model of computation, we only count 2-variable multiplications, and other operations are assumed to be free. It is motivated by the fact that 2-variable multiplication is often more expensive to compute than other operations and by the practicality of algorithms minimizing the multiplications, such as Karatsuba's and Strassen's. In particular when $\mathcal{A}$ is a finite dimensional algebra over $\mathbf{k}$, we define the bilinear complexity of $\mathcal{A}$ as $\mu(\mathcal{A}/\mathbf{k}) = \mu(m_\mathcal{A})$ where $m_\mathcal{A} : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ is the multiplication map in $\mathcal{A}$ seen as a $\mathbf{k}$-bilinear map.

Let $\mathbf{k}^{2\times2}$ be the algebra of 2×2 matrices over $\mathbf{k}$. We know thanks to Strassen's algorithm that

$$\mu(\mathbf{k}^{2\times2}/\mathbf{k}) \leq 7.$$

In fact, this is optimal [Win71, Theorem 3.1], so we have exactly $\mu(\mathbf{k}^{2\times2}/\mathbf{k}) = 7$. In general, it seems to be hard to find the bilinear complexity of a given algebra, for example the bilinear

complexity of $\mathbf{k}^{3\times 3}$ is not known. In the litterature, work has been done both to algorithmically find the bilinear complexity of small algebras [BDEZ12, Cov19] and to understand how the bilinear complexity asymptotically grows [CC88, BPR+21]. In 1988, Chudnovsky and Chudnovsky proposed a new method using evaluation-interpolation on curves, leading to the proof that the bilinear complexity of an extension field $\mathbb{F}_{q^k}/\mathbb{F}_q$ is linear in the degree k of the extension [Bal99]. We present this method in Section 3.2.1.

Bilinear formulas. We can precisely define bilinear complexity with *bilinear formulas*. We also sometimes use the terms *bilinear decomposition*, or *bilinear algorithm*, but it is really the same notion. For any vector space V, we let $V^\vee$ be its dual space, *i.e.* the vector space of $\mathbf{k}$-linear forms on V.

Definition 3.1.1 (Bilinear formula)**.** Let V_1, V_2 and W be three finite dimensional vector spaces over $\mathbf{k}$ and

$$\Phi : V_1 \times V_2 \to W$$

a bilinear map. A *bilinear fomula*, or *bilinear decomposition*, or *bilinear algorithm* of length n for Φ is a collection of $2n$ linear forms $\varphi_1, \ldots, \varphi_n \in V_1^\vee$ and $\psi_1, \ldots, \psi_n \in V_2^\vee$, and n vectors $w_1, \ldots, w_n$ in W such that for all $x \in V_1$ and $y \in V_2$, we have

$$\Phi(x, y) = \sum_{j=1}^{n} \varphi_j(x)\psi_j(y)w_j.$$

Let V_1 and V_2 be $\mathbf{k}$-vector spaces of respective dimensions l and m and let

$$x = (x_1, \ldots, x_l) \in V_1$$

and

$$y = (y_1, \ldots, y_m) \in V_2$$

be two vectors. A bilinear formula of length n for a bilinear map Φ is essentially a way of computing $\Phi(x, y)$ using a number n of 2-variable multiplications in $\mathbf{k}$. Indeed, for each $1 \leq j \leq n$ we have

$$\varphi_j(x) = \sum_{i=1}^{l} a_{i,j}x_i$$

and

$$\psi_j(y) = \sum_{i=1}^{m} b_{i,j}y_i$$

where the elements $a_{i,j}$ and $b_{i,j}$ are *constants* depending only on Φ. Thus the evaluation

$$\varphi_j(x)\psi_j(y)$$

only requires one 2-variable multiplication. The vectors $w_j \in W$ are also constants depending only on Φ, so we still need one 2-variable multiplication to compute

$$\varphi_j(x)\psi_j(y)w_j.$$

We said that the bilinear complexity measures the minimal number of 2-multiplications needed to compute a bilinear map Φ. Knowing that a bilinear formula of length n for Φ implies that we can compute Φ with the same number n of 2-variable multiplications, the definition of bilinear complexity follows.

Definition 3.1.2 (Bilinear complexity)**.** Let V_1, V_2 and W be three finite dimensional vector spaces over $\mathbf{k}$ and

$$\Phi : V_1 \times V_2 \to W$$

a bilinear map. The *bilinear complexity*

$$\mu(\Phi)$$

of Φ is the minimal length n of a bilinear formula for Φ.

Equivalently, we can define the bilinear complexity as the rank of the tensor in

$$V_1^\vee \otimes V_2^\vee \otimes W$$

corresponding to Φ [Ran12], as shown in Example 3.1.3.

Example 3.1.3. Let $\mathbf{k} = \mathbb{F}_2$ and $V_1 = V_2 = W = (\mathbb{F}_2)^2$. We consider the bilinear map Φ that follows.

$$\begin{array}{rccc} \Phi : & (\mathbb{F}_2)^2 \times (\mathbb{F}_2)^2 & \to & (\mathbb{F}_2)^2 \\ & ((x_0, x_1), (y_0, y_1)) & \mapsto & (x_0y_0 + x_1y_1, x_0y_1 + x_1y_0 + x_1y_1) \end{array}$$

Let $e_0 = (1, 0)$ and $e_1 = (0, 1)$ the vectors of the canonical basis of $(\mathbb{F}_2)^2$, and let $e_0^\vee$ and $e_1^\vee$ the vectors of the dual basis, *i.e.* the linear forms $e_0^\vee$ and $e_1^\vee$ are given by

$$\begin{array}{rccc} e_0^\vee : & (\mathbb{F}_2)^2 & \to & \mathbb{F}_2 \\ & (x_0, x_1) & \mapsto & x_0 \end{array}$$

and

$$\begin{array}{rccc} e_1^\vee : & (\mathbb{F}_2)^2 & \to & \mathbb{F}_2 \\ & (x_0, x_1) & \mapsto & x_1 \end{array}.$$

Let $x = (x_0, x_1)$ and $y = (y_0, y_1)$, we have

$$\begin{aligned} \Phi(x, y) &= (x_0y_0 + x_1y_1, x_0y_1 + x_1y_0 + x_1y_1) \\ &= (x_0y_0, x_0y_0) + (x_1y_1, 0) + (0, (x_0 + x_1)(y_0 + y_1)) \\ &= e_0^\vee(x)e_0^\vee(y)(e_0 + e_1) + e_1^\vee(x)e_1^\vee(y)e_0 + (e_0^\vee + e_1^\vee)(x)(e_0^\vee + e_1^\vee)(y)e_1. \end{aligned}$$

This last line is a bilinear formula of length 3 for Φ, and we can check that no formula of length 2 exists. Therefore the bilinear complexity of Φ is

$$\mu(\Phi) = 3.$$

Equivalently, the tensor in $((\mathbb{F}_2)^2)^\vee \otimes ((\mathbb{F}_2)^2)^\vee \otimes (\mathbb{F}_2)^2$ corresponding to Φ is

$$\begin{aligned} \widetilde{\Phi} &= e_0^\vee \otimes e_0^\vee \otimes e_0 + e_1^\vee \otimes e_1^\vee \otimes e_0 + e_0^\vee \otimes e_1^\vee \otimes e_1 + e_1^\vee \otimes e_0^\vee \otimes e_1 + e_1^\vee \otimes e_1^\vee \otimes e_1 \\ &= e_0^\vee \otimes e_0^\vee \otimes (e_0 + e_1) + e_1^\vee \otimes e_1^\vee \otimes e_0 + (e_0^\vee + e_1^\vee) \otimes (e_0^\vee + e_1^\vee) \otimes e_1, \end{aligned}$$

and we can check that no smaller decomposition of $\widetilde{\Phi}$ into a sum of simple tensors $a \otimes b \otimes c$ exists, so we also see that the rank of the tensor $\widetilde{\Phi}$ is 3.

When the spaces V_1 and V_2 are equal

$$V_1 = V_2 = V$$

the bilinear maps

$$\Phi : V \times V \to W$$

can be symmetric, *i.e.* they can verify that, for all $x, y \in V$

$$\Phi(x, y) = \Phi(y, x).$$

In that case, it is natural to investigate the existence and the length of *symmetric bilinear formulas*, *i.e.* bilinear formulas where the linear forms φ_j and ψ_j are equal, for all j. From an algorithmtic point of view, it should also be easier to find all such formulas because the search space is smaller. It is also easier to represent such formulas because we only need to store n linear forms instead of $2n$.

Definition 3.1.4 (Symmetric bilinear formula)**.** Let V and W be two finite dimensional vector spaces over $\mathbf{k}$ and

$$\Phi : V \times V \to W$$

a symmetric bilinear map. A *symmetric bilinear fomula*, or *symmetric bilinear decomposition*, or *symmetric bilinear algorithm* of length n for Φ is a collection of n linear forms $\varphi_1, \ldots, \varphi_n \in V^\vee$ and n vectors $w_1, \ldots, w_n$ in W such that for all $x, y \in V$, we have

$$\Phi(x, y) = \sum_{j=1}^{n} \varphi_j(x)\varphi_j(y)w_j.$$

Definition 3.1.5 (Symmetric bilinear complexity)**.** Let V and W be two finite dimensional vector spaces over $\mathbf{k}$ and

$$\Phi : V \times V \to W$$

a bilinear map. The *symmetric bilinear complexity*

$$\mu^{\mathrm{sym}}(\Phi)$$

of Φ is the minimal length n of a symmetric bilinear formula for Φ.

In other words, a symmetric bilinear formula is a bilinear formula where the domain spaces are equal: $V_1 = V_2$; and such that for all $1 \leq j \leq n$, the linear forms $\varphi_j = \psi_j$ are equal too. Note that it is not clear from Definition 3.1.4 that a *symmetric* bilinear formula always exists for symmetric bilinear maps, but it is indeed true [Ran12, Lemma 1.6], thus Definition 3.1.5 makes sense. The formula obtained in Example 3.1.3 is an example of bilinear formula that is also a symmetric bilinear formula, therefore the symmetric bilinear complexity is the same as the bilinear complexity in that case. We are particularly interested in algebras $\mathcal{A}$ of the form

$$\mathcal{A} = \mathbb{F}_{q^k}[T]/(T^l)$$

and for that reason we introduce a special notation for the bilinear complexity of those algebras

$$\mu_q(k, l) = \mu(\mathcal{A}/\mathbf{k}).$$

Among these algebras, the case $l = 1$, where the algebra $\mathcal{A}$ is a finite field extension of $\mathbb{F}_q$ of degree k also plays a special role, so we define

$$\mu_q(k) = \mu_q(k, 1).$$

Because these algebras are all commutative, the product

$$\begin{array}{rccc} m_{\mathcal{A}}: & \mathcal{A}\times\mathcal{A} & \to & \mathcal{A} \\ & (x,y) & \mapsto & xy \end{array}$$

is a symmetric bilinear map, and we define the symmetric bilinear complexity of the algebra $\mathcal{A}$ as the symmetric bilinear complexity of $m_{\mathcal{A}}$

$$\mu^{\mathrm{sym}}(\mathcal{A}) = \mu^{\mathrm{sym}}(m_{\mathcal{A}}).$$

We also define the quantities

$$\mu_q^{\mathrm{sym}}(k,l)$$

and

$$\mu_q^{\mathrm{sym}}(k)$$

the same way it was done for the usual bilinear complexity. Since a symmetric bilinear formula is in particular a bilinear formula, we have for all $k \geq 1$ and $l \geq 1$

$$\mu_q(k,l) \leq \mu_q^{\mathrm{sym}}(k,l).$$

In the other direction, we know ([SL84, Theorem 1] or [Ran12, Lemma 1.6]) that when the characteristic of $\mathcal{A}$ is not 2, or equivalently when k is not a power of 2, we have

$$\mu_q^{\mathrm{sym}}(k,l) \leq 2\mu_q(k,l).$$

Finally, we have no example of algebra $\mathcal{A} = \mathbb{F}_{q^k}[T]/(T^l)$ where the quantities $\mu_q(k,l)$ and $\mu_q^{\mathrm{sym}}(k,l)$ are different when $q \geq 3$.

3.2 Chudnovsky-Chudnovsky algorithm

Chudnovsky and Chudnovsky's algorithm is based on evaluation-interpolation on algebraic curves, we thus begin by presenting this principle.

3.2.1 Evaluation - Interpolation

Let $P \in \mathbf{k}[x]$ be a polynomial with coefficients in a finite field $\mathbf{k}$. The evaluation-interpolation strategy is based on two facts:

- a polynomial of degree n can be described by its values at $n+1$ points and reconstructed via *interpolation*;
- the *evaluation* map at some point $a \in \mathbf{k}$ is a homomorphism of rings from $\mathbf{k}[x]$ to $\mathbf{k}$.

Interpolation. The fact that a polynomial $P \in \mathbf{k}[x]$ of degree n is uniquely determined by its values at $n+1$ (different) points in $\mathbf{k}$ follows from the fact that a nonzero polynomial of degree n with coefficients in $\mathbf{k}$ has up to n roots. This gives us the *uniqueness* of the polynomial. As for the *existence*, it follows from the Lagrange interpolation. Let $x_1, \ldots, x_{n+1} \in \mathbf{k}$ be $n+1$ points in $\mathbf{k}$ and $y_1, \ldots, y_{n+1}$ the corresponding evaluation values, such that

$$\forall j \in \{1, \ldots, n+1\}, \; y_j = P(x_j).$$

Let

$$L_j = \prod_{i \neq j} \frac{x - x_i}{x_j - x_i},$$

we then have $L_j(x_i) = \delta_{i,j}$ with

$$\delta_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

the Kronecker symbol. Now, the polynomial

$$P = \sum_{j=1}^{n+1} y_j L_j$$

meets all the evaluation conditions and is the sum of polynomials of degree n so P is of degree at most n.

Evaluation. Let $P, Q \in \mathbf{k}[x]$ be two polynomials with coefficients in $\mathbf{k}$ and $a \in \mathbf{k}$, then we have

$$(P +_{\mathbf{k}[x]} Q)(a) = P(a) +_{\mathbf{k}} Q(a)$$

and

$$(P \times_{\mathbf{k}[x]} Q)(a) = P(a) \times_{\mathbf{k}} Q(a),$$

where $+_{\mathbf{k}[x]}, \times_{\mathbf{k}[x]}$ (resp. $+_{\mathbf{k}}, \times_{\mathbf{k}}$) are the addition and multiplication operations in $\mathbf{k}[x]$ (resp. $\mathbf{k}$). In other words, the map

$$\begin{array}{rccc} \mathrm{ev}_a : & \mathbf{k}[x] & \to & \mathbf{k} \\ & P & \mapsto & P(a) \end{array}$$

is a homomorphism of rings from $\mathbf{k}[x]$ to $\mathbf{k}$.

We are used to represent polynomials by their coefficients, but these two facts suggest that we can also represent polynomials by their values at some points. With this representation, adding two polynomials is done by adding the values, which is done with linear algebraic complexity, the same as with the coefficient representation. But the multiplication of polynomials is also obtained via the multiplication of the values, which is linear again and better than the quadratic complexity obtained with the usual multiplication formula for the coefficients. An important problem is then to be able to change between representations at a small cost, this is done using well-chosen points of evaluation and this strategy is known under the name of Fast Fourier Transform (FFT) [SS71, VZGG13, Chapter 8]. Let $P, Q \in \mathbf{k}[x]$ be two polynomials with coefficients in $\mathbf{k}$ represented by their coefficients, such that $\deg(PQ) = n - 1$. In order to multiply P and Q we need at least n points in $\mathbf{k}$ and the evaluation-interpolation strategy consists in 3 steps:

1. evaluation of P and Q at n points $a_1, \ldots, a_n$;
2. coordinate-wise multiplication;
3. interpolation to reconstruct the product PQ.

When there are not enough points in $\mathbf{k}$ to use this method, instead of evaluating on points of $\mathbf{k}$, we can evaluate the polynomials on points of algebraic curves over $\mathbf{k}$ with enough points. As a first example, we can interpret Karatsuba's algorithm as an evaluation-interpolation scheme on the projective line $\mathbb{P}^1(\mathbf{k})$. Let

$$P = a_1 x + a_0$$

and

$$Q = b_1 x + b_0,$$

then

$$c_0 = \mathrm{ev}_0(P)\mathrm{ev}_0(Q) = a_0 b_0$$

is obtained via evaluation at 0,

$$c_1 = \mathrm{ev}_1(P)\mathrm{ev}_1(Q) = (a_0 + a_1)(b_0 + b_1)$$

is obtained via evaluation at 1, and

$$c_\infty = \mathrm{ev}_\infty(P)\mathrm{ev}_\infty(Q) = a_1 b_1$$

is obtained via evaluation at the point at infinity, where the evaluation at infinity ev_∞ is the function mapping a polynomial to its leading coefficient. This strategy can be generalized to curves (or their function fields) more complex than $\mathbb{P}^1(\mathbf{k})$, as was done by Chudnovsky and Chudnovsky in 1988 [CC88].

3.2.2 Asymptotic complexity

In 1988, Chudnovsky and Chudnovsky [CC88] extended the idea of polynomial interpolation to interpolation on rational places, *i.e.* places of degree 1, of a function field. It led to an algorithm for the finite field product with an asymptotically linear complexity in the extension degree. We first present the historical theorem in [CC88].

Theorem 3.2.1. *Let F be a function field over $\mathbb{F}_q$. Assume there exist a place $Q \in \mathbb{P}_F$ of F of degree k, $P_1, \ldots, P_n \in \mathbb{P}_F$ places of F of degree 1, and a divisor $D \in \mathcal{D}_F$ of F such that the places Q and $P_1, \ldots, P_n$ are not in the support of D and such that the following conditions hold.*

(i) The evaluation map

$$\begin{array}{rccc} \mathrm{ev}_{Q,D}: & L(D) & \to & \mathbb{F}_{q^k} \\ & f & \mapsto & f(Q) \end{array}$$

is surjective.

(ii) The evaluation map

$$\begin{array}{rccc} \mathrm{ev}_{\mathcal{P},2D}: & L(2D) & \to & (\mathbb{F}_q)^n \\ & h & \mapsto & (h(P_1), \ldots, h(P_n)) \end{array}$$

is injective.

Then the product in the extension field

$$\mathbb{F}_{q^k}/\mathbb{F}_q$$

admits a symmetric formula of length n, i.e. *we have $\mu_q^{sym}(k) \leq n$.*

Theorem 3.2.1 can be interpreted in terms of evaluation and interpolation. Condition (i) ensures that any element $x \in \mathbb{F}_{q^k}$ can be represented by a function $f_x \in L(D)$. Given two elements $x, y \in \mathbb{F}_{q^k}$ that we want to multiply, we thus represent them as functions $f_x, f_y \in L(D)$ and we *evaluate* these functions at the n places $P_1, \ldots, P_n$ of degree 1. We obtain two elements

$$a_x = (f_x(P_1), \ldots, f_x(P_n))$$

and

$$a_y = (f_y(P_1), \ldots, f_y(P_n))$$

that we multiply coefficient-wise in order to obtain

$$a_{xy} = (f_x(P_1)f_y(P_1), \ldots, f_x(P_n)f_y(P_n)).$$

Now, the injectivity in Condition (ii) ensures that the element a_{xy}, *i.e.* the evaluations at the points $P_1, \ldots, P_n$, defines a unique function in $L(2D)$, that is in fact $f_x f_y$. Indeed, we see that the function $f_x f_y$ is in $L(2D)$ since f_x and f_y are each in $L(D)$, and we have

$$\mathrm{ev}_{\mathcal{P},2D}(f_x f_y) = a_{xy}.$$

Given the evaluations a_{xy}, we thus *interpolate*, *i.e.* reconstruct a function

$$g = f_x f_y \in L(2D)$$

that we finally evaluate at the place Q to recover

$$g(Q) = f_x(Q)f_y(Q) = xy.$$

This whole process costs n multiplications: those that appear when we compute the coefficient-wise multiplication of a_x and a_y. Details can be found in [CC88]. Another version of Theorem 3.2.1, generalized to the case of the multiplication of an arbitrary number $s \geq 2$ of variables, is also discussed in Proposition 4.3.2. One can also give sufficient numerical conditions [Bal98, Bal99] on the genus g of the function field F and the number n of places $P_1, \ldots, P_n$ of degree 1 to ensure that Conditions (i) and (ii) are met. The challenge is then to find function fields that meet the conditions: they must have many places of degree 1, while trying to maintain the genus at a minimum. Using for example [STV92] or [Pie12], we know that there exist suitable families of function fields, but it is quite arduous to find good asymptotic families. Following the pioneer idea of Chudnovsky and Chudnovsky and additionnal work from Shparlinski, Tfasman and Vlăduţ [STV92], Ballet was able to prove [Bal99] that the bilinear complexity of the multiplication in the finite field extension

$$\mathbb{F}_{q^k}/\mathbb{F}_q$$

is *linear* in the degree of the extension k. Theorem 3.2.1 was then generalized by Ballet and Rolland [BR04] and Cenk and Özbudak [CÖ10] in order to exploit places of higher degrees. Finally the most general version, allowing us to use asymmetric formulas, was proposed by Randriambololona in [Ran12]. All the historical and technical details can be found in the survey of Ballet, Chaumine, Pieltant, Rambaud, Randriambololona and Rolland [BPR+21].

3.3 Algorithmic searches in small dimension

The last results based on Chudnovsky and Chudnovsky's algorithm allow us to find one decomposition, and thus give us an (asymptotic) upper bound on the bilinear complexity of algebras

$$\mathcal{A} = \mathbb{F}_{q^k}[T]/(T^l).$$

When one wants to find the exact value of the bilinear complexity of a given bilinear map Φ, one can either find all bilinear formulas for Φ, or find a theoretical argument to directly find the bilinear complexity of Φ. The latter solution is often hard, and there also exist some in-between

approaches such as finding a bilinear formula of a given length and proving that no shorter formula could exist. Still, because it seems hard to directly find the bilinear complexity of a bilinear map, algorithms were developped to find bilinear formulas. These algorithms are essentially exhaustive searches, so they have an exponential complexity, but they also exploit the eventual symmetries in the definition of Φ to eliminate a lot of potential candidates along the way, in order to be as efficient as possible in practice. We first look at Barbulescu, Detrey, Estibals and Zimmerman's algorithm [BDEZ12].

3.3.1 Barbulescu, Detrey, Estibals and Zimmerman's algorithm

In 2012, Barbulescu, Detrey, Estibals and Zimmerman published a new framework to find bilinear formulas for arbitrary bilinear maps over finite fields. Let V_1, V_2 and W be three finite dimensional $\mathbf{k}$-vector spaces of respective dimensions l, m, and n, such that we have

$$V_1 \cong \mathbf{k}^l \qquad V_2 \cong \mathbf{k}^m \qquad W \cong \mathbf{k}^n.$$

Let

$$\Phi : V_1 \times V_2 \to W$$

be a bilinear map, and denote by $\mathcal{B}$ the space of bilinear *forms* from $V_1 \times V_2$ to $\mathbf{k}$. Let $\gamma \in \mathcal{B}$ a bilinear form, then if

$$x = (x_1, \ldots, x_l) \in V_1$$

and

$$y = (y_1, \ldots, y_m) \in V_2,$$

then γ is given by

$$\gamma(x, y) = \sum_{i=1}^{l} \sum_{j=1}^{m} \gamma_{i,j} x_i y_j.$$

Hence, $\mathcal{B}$ is a $\mathbf{k}$-vector space of dimension lm and we can see γ as a vector

$$\gamma = (\gamma_{1,1}, \ldots, \gamma_{l,m}).$$

Another interesting representation is to see γ as a $l \times m$ matrix

$$G = (\gamma_{i,j})_{i,j},$$

such that γ is given by

$$\gamma(x, y) = xGy^t$$

where y^t is the transpose of y and thus y^t is a column vector. We let γ_j be the j-th coordinate of Φ in W, such that

$$\Phi = (\gamma_1, \ldots, \gamma_n).$$

If $n = 1$, Φ is a bilinear form and its bilinear complexity is given by the rank of its matrix representation. The reason is that the rank of a matrix is invariant by change of basis, a rank r matrix is the sum of r matrices of rank 1, and a rank 1 matrix corresponds to a bilinear form that can be evaluated using only one 2-variable multiplication. Indeed, let A be a $l \times m$ matrix of rank 1, we know that there exist a nonzero vector

$$b = (b_1, \ldots, b_m) \in \mathbf{k}^m$$

such that the rows $(r_j)_{1\leq j\leq m}$ of A, seen as vectors $\mathbf{k}^m$, are all multiples of b, *i.e.* for all $1 \leq j \leq l$, we have

$$r_j = a_j b$$

with $a_j \in \mathbf{k}$, and so

$$A = \begin{bmatrix} a_1 b \\ \vdots \\ a_l b \end{bmatrix}.$$

Let γ be the bilinear form represented by A, $x \in \mathbf{k}^l$ and $y \in \mathbf{k}^m$, then

$$\begin{aligned} \gamma(x,y) &= xAy^t \\ &= \sum_{i=1}^{l}(x_i \sum_{j=1}^{m} a_i b_j y_j) \\ &= (\sum_{i=1}^{l} a_i x_i) \times (\sum_{j=1}^{m} b_j y_j). \end{aligned}$$

The elements $a_1, \ldots, a_l$ and $b_1, \ldots, b_m$ only depend on γ (or equivalently, on A), so they are constants, thus evaluating γ only requires one 2-variable multiplication. When $n > 1$, *i.e.* Φ is not a bilinear form, there is no similar way of knowing the bilinear complexity of

$$\Phi = (\gamma_1, \ldots, \gamma_n).$$

Nonetheless, Barbulescu *et al.* presented an algorithm to find bilinear formulas for Φ. The representation that is used for this algorithm is the vectorial one, and the subspace

$$V = \mathrm{Span}\,\{\gamma_1, \ldots, \gamma_n\} \subset \mathcal{B}$$

plays a central role. The general idea is to find a generating family of V composed of rank 1 bilinear forms.

Proposition 3.3.1. *Let V_1, V_2, and W be three finite $\mathbf{k}$-vector spaces and $\Phi : V_1 \times V_2 \to W$ a bilinear map such that*

$$\Phi = (\gamma_1, \ldots, \gamma_n).$$

Let

$$V = \mathrm{Span}\,\{\gamma_1, \ldots, \gamma_n\}$$

and let

$$\mathcal{F} = \{\phi_1, \ldots, \phi_t\}$$

be a generating family of V composed of rank 1 bilinear forms. Then there exists a bilinear formula of length t for Φ.

Proof. Let

$$\{e_1, \ldots, e_n\}$$

be a basis of W. For each $1 \leq j \leq n$, let

$$\gamma_j = \sum_{i=1}^{t} a_{i,j} \phi_i$$

be a decomposition of γ_j in the generating family $\mathcal{F}$. It follows that

$$\begin{aligned}\Phi &= \sum_{j=1}^{n} \gamma_j e_j \\ &= \sum_{j=1}^{n} (\sum_{i=1}^{t} a_{i,j}\phi_i) e_j \\ &= \sum_{i=1}^{t} \phi_i (\sum_{j=1}^{n} a_{i,j} e_j) \\ &= \sum_{i=1}^{t} \phi_i w_i\end{aligned}$$

with, for all $1 \leq i \leq t$,

$$w_i = \sum_{j=1}^{n} a_{i,j} e_j$$

a (constant) vector in W and ϕ_i is a rank 1 bilinear form, so that it can be evaluated using only one 2-variable multiplication in $\mathbf{k}$. Therefore we have a bilinear formula of length t for Φ. □

If Φ is a symmetric bilinear map, the same strategy can be used with a basis composed of symmetric bilinear forms of rank 1. Now the question is how to find such generating family $\mathcal{F}$. Let

$$\mathcal{G} = \{\phi \in \mathcal{B} \,|\, \phi \text{ is of rank } 1\}$$

be the set of bilinear forms of rank 1. In order to find bilinear formulas of length t, a naive solution is to exhaustively search for elements

$$g_1, \ldots, g_t \in \mathcal{G}$$

such that

$$V \subseteq \text{Span}\{g_1, \ldots, g_t\}. \tag{3.1}$$

There are

$$\binom{\#\mathcal{G}}{t}$$

possible choices for the t-uple $(g_1, \ldots, g_t)$, and for each t-uple we have to test the subspace condition (3.1). For the sake of simplicity, we consider the *combinatorial complexity* of this problem, *i.e.* we consider that all matrix operations have constant complexity (*e.g.* computing the dimension of a vector space, checking if a vector is in a vector space). Thus the combinatorial complexity of this strategy is $\binom{\#\mathcal{G}}{t}$. Unfortunately, two different t-uples of generators in $\mathcal{G}$ can span the same vector space, and so the naive algorithm is non-optimal. Barbulescu *et al.* attack this problem by looking directly at subspaces W such that

$$V \subseteq W.$$

More precisely, if we want to find a length t formula, we search for spaces W such that

(i) $V \subseteq W$;

(ii) $\operatorname{Span}(W \cap \mathcal{G}) = W$, *i.e.* W is generated by rank 1 bilinear forms;

(iii) $\dim W = t$, *i.e.* t generators of $\mathcal{G}$ are needed.

We remark thanks to (i) that V is contained in each space W that we are looking for. Thus, we can search for spaces W by adding elements to V. Of course, there are lots of spaces verifying (i) alone, but this condition, together with (ii), is in fact equivalent to

(ii') $\exists\, \mathcal{W} \subset \mathcal{G}$ such that $W = V \oplus \operatorname{Span} \mathcal{W}$,

therefore, we need only to add elements from $\mathcal{G}$ to V.

Lemma 3.3.2. *All spaces W that verify (i) and (ii) also verify (ii').*

Proof. Assume that conditions (i) and (ii) are satisfied, let

$$l = \dim V$$

and

$$m = \dim W.$$

Let B a basis of W composed of elements of G, *i.e.* rank 1 bilinear forms. We construct a family $(\mathcal{W}_j)_{0 \leq j \leq m-l}$ of subsets $\mathcal{W}_j \subset \mathcal{G}$ such that for all $0 \leq j \leq m - l$

$$\dim(V \oplus \operatorname{Span} \mathcal{W}_j) = l + j$$

and

$$V \oplus \operatorname{Span} \mathcal{W}_j \subseteq W.$$

We let $\mathcal{W}_0 = \emptyset$ be the nullspace, and we inductively construct $\mathcal{W}_j$ for $1 \leq j \leq m - l$. Assume that $\dim(V \oplus \operatorname{Span} \mathcal{W}_{j-1}) = l + j - 1$ and $V \oplus \operatorname{Span} \mathcal{W}_{j-1} \subseteq W$. We choose an element φ in B such that

$$\varphi \notin V \oplus \operatorname{Span} \mathcal{W}_{j-1}.$$

Such an element exists, otherwise we would have

$$V \oplus \operatorname{Span} \mathcal{W}_{j-1} = W$$

and so

$$\dim(V \oplus \operatorname{Span} \mathcal{W}_{j-1}) = l + j - 1 = \dim W = m.$$

Since $j \leq m - l$, it would follow that

$$m = l + j - 1 \leq m - 1,$$

a contradiction. Hence, we define

$$\mathcal{W}_j = \mathcal{W}_{j-1} \cup \{\varphi\},$$

and we have

$$\dim(V \oplus \operatorname{Span} \mathcal{W}_j) = l + j$$

and

$$V \oplus \operatorname{Span} \mathcal{W}_j \subseteq W.$$

We take the set $\mathcal{W} = \mathcal{W}_{m-l}$ and we have the desired property. □

Lemma 3.3.2 essentially tells us that the combinatorial complexity of finding a length t formula is

$$\binom{\#\mathcal{G}}{t - \dim V}$$

because we can take advantage of the fact that we already know that a solution space W verifies $V \subseteq W$. We introduce some more notation: we let

$$\mathcal{S}_t = \{W \subseteq \mathcal{B} \mid \mathrm{Span}(W \cap \mathcal{G}) = W \text{ and } \dim W = t\}$$

be the set of vector subspaces of $\mathcal{B}$ of dimension t that are generated by rank 1 bilinear forms. We also let

$$\mathcal{S}_t(V) = \{W \in \mathcal{S}_t \mid V \subseteq W\}$$

the subspaces in $\mathcal{S}_t$ that also contain V. With this new terminology, if we want to compute formulas of length t for

$$\Phi = (\gamma_1, \ldots, \gamma_n),$$

our goal is now to compute $\mathcal{S}_t(V)$ with

$$V = \mathrm{Span}(\{\gamma_1, \ldots, \gamma_n\}).$$

In order to effectively compute these sets, we use Algorithm 1 to check if a vector space is generated by rank 1 bilinear forms. The algorithm following Lemma 3.3.2 is described in Algorithm 2.

Algorithm 1 HasRankOneBasis

Input: $V \subseteq \mathcal{B}$ a subspace of $\mathcal{B}$ of dimension l
Output: A boolean indicating if $V \in \mathcal{S}_l$

```
1: H ← Span(V ∩ G)
2: if dim H = dim V then
3:     return true
4: else
5:     return false
6: end if
```

There is another advantage of Algorithm 2 compared to the naive algorithm. We see in Line 7 and Line 14 that it takes into account the equivalence relation "modulo V", *i.e.* if there are two elements ϕ and ϕ' of $\mathcal{G}$ such that

$$V \oplus \mathrm{Span}\,\phi = V \oplus \mathrm{Span}\,\phi',$$

only one representative is explored in the tree of recursive calls. As an example, if

$$\mathcal{G} = \{\phi_1, \phi_2, \phi_3, \phi_4\}$$

contains 4 rank 1 bilinear forms, and if $t - \dim V = 3$, the tree of recursive calls would generically (*e.g.* if the bilinear forms ϕ_j are independent modulo V) look like Figure 3.2, where we let

$$W_i = \mathrm{Span}(\{\phi_i\}),$$

$$W_{i,j} = \mathrm{Span}(\{\phi_i, \phi_j\}),$$

Algorithm 2 Barbulescu, Detrey, Estibals, Zimmerman

Input: $V \subseteq \mathcal{B}$ a subspace of $\mathcal{B}$.; $t \in \mathbb{N}$ an integer

Output: A list of formulas of length t for Φ.

```
1: function ExpandSubspace(X, H, d, t)
2:     if d = t and dim X = t and HasRankOneBasis(X) then
3:         return {X}
4:     else
5:         S ← ∅
6:         for i = 1 to #H do                                   ▷ H = {φ_1, ..., φ_v}
7:             H' ← {φ_{i+1}, ..., φ_v}  mod φ_i                ▷ Gaussian elimination modulo φ_i
8:             S ← S ∪ ExpandSubspace(X ⊕ Span(φ_i), H', d + 1, t)
9:         end for
10:        return S
11:    end if
12: end function
13: V = Span({γ_1, ..., γ_n})
14: return ExpandSubspace(V, G  mod V, dim V, t)    ▷ Gaussian reduction of G modulo a basis of V
```

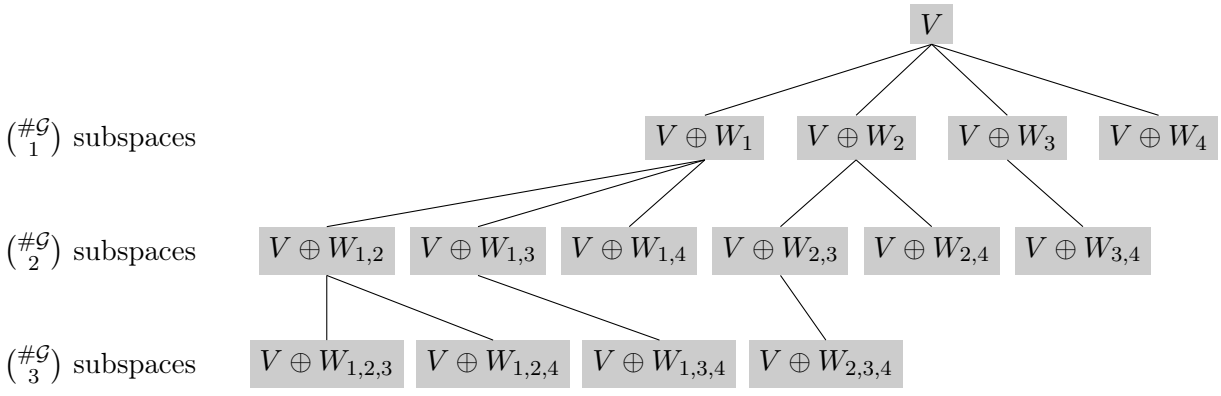


Figure 3.2: Tree of recursive calls in Algorithm 2, with $\#\mathcal{G} = 4$ and $t - \dim V = 3$.

and

$$W_{i,j,k} = \mathrm{Span}(\{\phi_i, \phi_j, \phi_k\}).$$

Algorithm 2 can also be adapted to find *symmetric* formulas: instead of searching in the set $\mathcal{G}$, we search in the set

$$\mathcal{G}_{\mathrm{sym}} = \{\phi \in \mathcal{B} \,|\, \phi \text{ is symmetric and is of rank } 1\}\,.$$

Since

$$\#\mathcal{G}_{\mathrm{sym}} = \sqrt{\#\mathcal{G}},$$

the complexity of Algorithm 2 adapted to the symmetric case is naturally better. Although not initially published in [BDEZ12], Barbulescu, Detrey, Estibals and Zimmerman improved on their algorithms by using symmetries in the definition of the subspaces of $\mathcal{B}$. Their strategy was described in [Cov19], as well as further improvements from Covanov, exploiting even more symmetries. These ideas inspired our work on other kinds of decompositions introduced in Chapter 4, for which we provide an *ad hoc* search algorithm.

Chapter 4

Hypersymmetric bilinear complexity

In Chapter 3, we have seen the notions of bilinear complexity and symmetric bilinear complexity. We now investigate even stronger notions of symmetry, yielding very short representations of a bilinear map.

Contents

4.1 Symmetric and hypersymmetric fomulas 58
4.1.1 Generalization to multilinear maps 58
4.1.2 Trisymmetric and hypersymmetric complexity 60
4.1.3 Galois invariance 63
4.1.4 Multiplication formulas in algebras 65
4.2 Algorithmic search in small dimension 65
4.2.1 General algorithm description 66
4.2.2 Implementation 74
4.2.3 Universal formulas 77
4.3 Asymptotic complexities 80

4.1 Symmetric and hypersymmetric fomulas

Let $\mathbf{k}$ be a finite field, V_1, V_2 and W three finite-dimensional $\mathbf{k}$-vector spaces and

$$\Phi : V_1 \times V_2 \to W$$

a bilinear map. Recall Definition 3.1.1:

$$\Phi(x, y) = \sum_{j=1}^{t} \varphi_j(x)\psi_j(y)w_j,$$

where for all $1 \leq j \leq t$, $\varphi_j \in V_1^\vee$ and $\phi_j \in V_2^\vee$ are linear forms and $w_j \in W$ is a vector, is called a *bilinear formula* of length t. If the spaces V_1 and V_2 are equal and if the bilinear map Φ is symmetric, *i.e.* if for all $x, y \in V$

$$\Phi(x, y) = \Phi(y, x),$$

we can investigate the existence of formulas satisfying the same condition of symmetry, *i.e.* formulas where for all $1 \leq j \leq t$, $\varphi_j = \psi_j$, resulting in a *symmetric* bilinear formula:

$$\Phi(x, y) = \sum_{j=1}^{t} \varphi_j(x)\varphi_j(y)w_j.$$

In fact, we can define other interesting types of symmetries, but it is useful to first generalize the notions that we saw in Chapter 3 to higher dimensions.

4.1.1 Generalization to multilinear maps

The definitions of bilinear formula and bilinear complexity are not limited to the bilinear case and can be generalized to arbitrary dimension. These general definitions will be used in Section 4.1.2 to define *hypersymmetric* complexity.

Definition 4.1.1 (Multilinear formula)**.** Let $V_1, V_2, \ldots, V_s$ and W be $s+1$ finite-dimensional $\mathbf{k}$-vector spaces and

$$\Phi : V_1 \times V_2 \times \cdots \times V_s \to W$$

an s-linear map. A *multilinear formula*, or *multilinear decomposition*, or *multilinear algorithm* of length t for Φ is a collection of $s \times t$ linear forms $\varphi_1^{(1)}, \varphi_2^{(1)}, \ldots, \varphi_t^{(1)} \in V_1^\vee$ up to $\varphi_1^{(s)}, \varphi_2^{(s)}, \ldots, \varphi_t^{(s)} \in V_s^\vee$ and t vectors $w_1, \ldots, w_t$, such that for all $x_1 \in V_1, \ldots, x_s \in V_s$, we have

$$\Phi(x_1, \ldots, x_s) = \sum_{j=1}^{t} \varphi_j^{(1)}(x_1) \ldots \varphi_j^{(s)}(x_s)w_j.$$

Definition 4.1.2 (Multilinear complexity)**.** Let $V_1, V_2, \ldots, V_s$ and W be $s+1$ finite-dimensional $\mathbf{k}$-vector spaces and

$$\Phi : V_1 \times V_2 \times \cdots \times V_s \to W$$

an s-linear map. The *multilinear complexity* $\mu(\Phi)$ of Φ is the minimal length t of a multilinear formula for Φ.

As in the case of bilinear complexity, the multilinear complexity $\mu(\Phi)$ of a multilinear map Φ can also be defined as the rank of the tensor in

$$V_1^\vee \otimes \cdots \otimes V_s^\vee \otimes W$$

corresponding to Φ, see Example 3.1.3 for an illustration of this correspondence in the bilinear case. In the case where

$$V_1 = V_2 = \cdots = V_s,$$

symmetric formulas and symmetric complexity can also be generalized when Φ is a *symmetric* multilinear map, *i.e.* when for all permutations $\sigma \in \mathfrak{S}_s$ and for all vectors $x_1, \ldots, x_s \in V$, we have

$$\Phi(x_1, \ldots, x_s) = \Phi(x_{\sigma(1)}, \ldots, x_{\sigma(s)}).$$

Definition 4.1.3 (Symmetric multilinear formula)**.** Let V and W be two finite-dimensional $\mathbf{k}$-vector spaces and

$$\Phi : \underbrace{V \times \cdots \times V}_{s \text{ times}} \to W$$

be a symmetric s-linear map. A *symmetric multilinear formula*, or *symmetric multilinear decomposition*, or *symmetric multilinear algorithm* of length t for Φ is a collection of t linear forms $\varphi_1, \varphi_2, \ldots, \varphi_t \in V^\vee$ and t vectors $w_1, \ldots, w_t$, such that for all $x_1, \ldots, x_s \in V$, we have

$$\Phi(x_1, \ldots, x_s) = \sum_{j=1}^{t} \varphi_j(x_1) \ldots \varphi_j(x_s) w_j.$$

Definition 4.1.4 (Symmetric multilinear complexity)**.** Let V and W be two finite-dimensional $\mathbf{k}$-vector spaces and

$$\Phi : \underbrace{V \times \cdots \times V}_{s \text{ times}} \to W$$

be a symmetric s-linear map. The *symmetric multilinear complexity* $\mu^{\text{sym}}(\Phi)$ of Φ is the minimal length t of a symmetric multilinear formula for Φ. If no such formula exists, we set

$$\mu^{\text{sym}}(\Phi) = \infty.$$

Contrary to the bilinear case, some symmetric multilinear maps do not admit a symmetric decomposition, but the problem of whether a symmetric multilinear map admits a symmetric multilinear formula is well understood and follows from Theorem 4.1.5.

Theorem 4.1.5 ([Ran15, Thm. A.7])**.** *Let $\Phi : V^s \to W$ be a s-linear map between finite dimensional vector spaces over $\mathbb{F}_q$. Then Φ admits a symmetric decomposition if and only if Φ is* Frobenius-symmetric*,* i.e. *if and only if it is symmetric and one of the following two conditions holds:*

- $s \leq q$
- $s \geq q+1$ *and for all $u, v, z_1, \ldots, z_{s-q-1}$ in V,*

$$\Phi(\underbrace{u, \ldots, u}_{q \text{ times}}, v, z_1, \ldots, z_{s-q-1}) = \Phi(u, \underbrace{v, \ldots, v}_{q \text{ times}}, z_1, \ldots, z_{s-q-1}).$$

4.1.2 Trisymmetric and hypersymmetric complexity

Under even stricter conditions, we can study the existence of even more symmetric formulas. These formulas allow us to describe a multilinear map with fewer elements, and thus give a compact definition of the map. Since the symmetry conditions are stronger there are fewer such formulas, and as a consequence the search space is smaller. Thus, we expect search algorithms to be faster, as was the case when using Barbulescu *et al.*'s algorithm (Algorithm 2) to find symmetric formulas. Let us define those "stricter conditions". Let

$$\Phi : V^s \to V$$

be an s-linear symmetric map, *i.e.* we additionally ask that $W = V$. We also assume that V has a non-degenerate symmetric bilinear form, that we write as a scalar product

$$\begin{array}{ccc} V \times V & \to & \mathbf{k} \\ (v, w) & \mapsto & \langle v, w \rangle . \end{array}$$

In that case, we know that the vector space V is isomorphic to its dual space $V^\vee$:

$$V \cong V^\vee,$$

i.e. for each linear form $\varphi \in V^\vee$, there exist a unique vector $a \in V$ such that for all $x \in V$, we have

$$\varphi(x) = \langle a, x \rangle .$$

Under these conditions, we can now write a symmetric formula for Φ as

$$\Phi(x_1, \ldots, x_s) = \sum_{j=1}^{t} \langle a_j, x_1 \rangle \ldots \langle a_j, x_s \rangle \, w_j.$$

where for all $1 \leq j \leq t$, $a_j \in V$ is a vector of V. As a consequence, we can also describe a symmetric formula for Φ as the data of vectors $(a_j)_{1 \leq j \leq t}$ and $(w_j)_{1 \leq j \leq t}$. In order to have an even more compact description of Φ, one can ask for the vectors w_j to be proportional to a_i, leading to the definition of hypersymmetric formula.

Definition 4.1.6 (Hypersymmetric formula)**.** Let V be a finite-dimensional $\mathbf{k}$-vector space equipped with a scalar product and

$$\Phi : \underbrace{V \times \cdots \times V}_{s \text{ times}} \to V$$

a symmetric s-linear map. A *hypersymmetric formula*, or *hypersymmetric decomposition*, or *hypersymmetric algorithm* of length t for Φ is a collection of t vectors $a_1, \ldots, a_t \in V$ and t scalars $\lambda_1, \ldots, \lambda_t \in \mathbf{k}$, such that for all $x_1, \ldots, x_s \in V$, we have

$$\Phi(x_1, \ldots, x_s) = \sum_{j=1}^{t} \lambda_j \langle a_j, x_1 \rangle \ldots \langle a_j, x_s \rangle \, a_j.$$

Definition 4.1.7 (Hypersymmetric complexity)**.** Let V be a finite-dimensional $\mathbf{k}$-vector space equipped with a scalar product and

$$\Phi : \underbrace{V \times \cdots \times V}_{s \text{ times}} \to V$$

a symmetric s-linear map. The *hypersymmetric complexity* $\mu^{\mathrm{hyp}}(\Phi)$ of Φ is the minimal length t of a hypersymmetric formula for Φ. If no such formula exists, we set

$$\mu^{\mathrm{hyp}}(\Phi) = \infty.$$

Example 4.1.8. We take the same case as in Example 3.1.3, but viewed a bit differently. Let $\mathbf{k} = \mathbb{F}_2$ and

$$V = \mathbb{F}_4 \cong \mathbb{F}_2[T]/(T^2 + T + 1) \cong \mathbb{F}_2(\zeta)$$

seen as a $\mathbb{F}_2$-vector space of dimension 2 using the base $(1, \zeta)$. The $\mathbb{F}_2$-bilinear map Φ that we consider is the product in $\mathbb{F}_4$:

$$\begin{array}{rccc} \Phi: & \mathbb{F}_4 \times \mathbb{F}_4 & \to & \mathbb{F}_4 \\ & (x, y) & \mapsto & xy. \end{array}$$

We also consider the non-degenerate symmetric bilinear form

$$\begin{array}{ccc} \mathbb{F}_4 \times \mathbb{F}_4 & \to & \mathbb{F}_2 \\ (v, w) & \mapsto & \mathrm{Tr}(vw), \end{array}$$

where Tr is the trace of the field extension $\mathbb{F}_4/\mathbb{F}_2$, and we write

$$\mathrm{Tr}(xy) = \langle x, y \rangle .$$

If $x = x_0 + x_1\zeta \in \mathbb{F}_4$ is an element in the extension field, we have $\mathrm{Tr}(x) = x_1$, and if $y = y_0 + y_1\zeta \in \mathbb{F}_4$ is another element, then their product is

$$xy = x_0y_0 + x_1y_1 + (x_0y_1 + x_1y_0 + x_1y_1)\zeta.$$

We also see that

$$\begin{cases} \langle 1, x \rangle \langle 1, y \rangle & = & x_1y_1 \\ \langle 1 + \zeta, x \rangle \langle 1 + \zeta, y \rangle & = & x_0y_0 \\ \langle \zeta, x \rangle \langle \zeta, y \rangle & = & (x_0 + x_1)(y_0 + y_1) \end{cases}$$

and thus we have

$$xy = \langle 1, x \rangle \langle 1, y \rangle \cdot 1 + \langle 1 + \zeta, x \rangle \langle 1 + \zeta, y \rangle \cdot (1 + \zeta) + \langle \zeta, x \rangle \langle \zeta, y \rangle \cdot \zeta.$$

This is an hypersymmetric formula of length 3, and we can prove that there are no formulas of length 2, so we have

$$\mu^{\mathrm{hyp}}(\Phi) = 3.$$

This is in fact the very same formula as in Example 3.1.3.

In order to investigate the existence of hypersymmetric decompositions, we remark that there is a natural link between hypersymmetric decompositions of the s-linear map

$$\Phi : V^s \to V$$

and symmetric decompositions of the $(s+1)$-linear form $\widetilde{\Phi}$ defined by

$$\begin{array}{rccc} \widetilde{\Phi}: & V^{s+1} & \to & \mathbf{k} \\ & (x_1, \ldots, x_{s+1}) & \mapsto & \langle \Phi(x_1, \ldots, x_s), x_{s+1} \rangle \end{array}$$

given by Lemma 4.1.9. Definition 4.1.10 follows from this correspondence.

Lemma 4.1.9. *Let V a $\mathbf{k}$-vector space and*

$$\Phi : V^s \to V$$

a symmetric s-linear map. Elements $(a_j)_{1\leq j\leq t}$ in V and scalars $(\lambda_j)_{1\leq j\leq t}$ in $\mathbf{k}$ define a hypersymmetric formula for the s-linear map Φ,

$$\Phi(x_1,\ldots,x_s) = \sum_{j=1}^{t} \lambda_j \langle a_j, x_1\rangle \cdots \langle a_j, x_s\rangle a_j,$$

if and only if they define a symmetric formula for the $(s+1)$-linear form $\widetilde{\Phi}$,

$$\widetilde{\Phi}(x_1,\ldots,x_s,x_{s+1}) = \sum_{j=1}^{t} \lambda_i \langle a_j, x_1\rangle \cdots \langle a_j, x_s\rangle \langle a_j, x_{s+1}\rangle .$$

Thus, Φ admits a hypersymmetric formula if and only if $\widetilde{\Phi}$ is Frobenius-symmetric (in the sense of Theorem 4.1.5), and we have

$$\mu^{hyp}(\Phi) = \mu^{sym}\left(\widetilde{\Phi}\right).$$

In particular, if $q \geq s+1$, then any hypersymmetric s-linear map over $\mathbb{F}_q$ admits a hypersymmetric formula.

Proof. Assume that Φ admits a hypersymmetric decomposition, such that for all $x_1,\ldots,x_s \in V$, we have

$$\Phi(x_1,\ldots,x_s) = \sum_{j=1}^{t} \lambda_i \langle a_j, x_1\rangle \cdots \langle a_j, x_t\rangle a_j,$$

then, by taking the scalar product with any x_{s+1}, we obtain

$$\langle \Phi(x_1,\ldots,x_s), x_{s+1}\rangle = \widetilde{\Phi}(x_1,\ldots,x_s,x_{s+1}) = \sum_{j=1}^{t} \lambda_i \langle a_j, x_1\rangle \cdots \langle a_j, x_s\rangle \langle a_j, x_{s+1}\rangle ,$$

which defines a symmetric decomposition for $\widetilde{\Phi}$. In the other direction, assume that $\widetilde{\Phi}$ admits a symmetric decomposition, such that for all $x_1,\ldots,x_{s+1} \in V$, we have

$$\widetilde{\Phi}(x_1,\ldots,x_s,x_{s+1}) = \sum_{j=1}^{t} \lambda_i \langle a_j, x_1\rangle \cdots \langle a_j, x_s\rangle \langle a_j, x_{s+1}\rangle .$$

It can also be written as

$$\langle \Phi(x_1,\ldots,x_s), x_{s+1}\rangle = \left\langle \sum_{j=1}^{t} \lambda_j \langle a_j, x_1\rangle \cdots \langle a_j, x_s\rangle a_j, x_{s+1} \right\rangle ,$$

so that we have

$$\left\langle \Phi(x_1,\ldots,x_s) - \sum_{j=1}^{t} \lambda_j \langle a_j, x_1\rangle \cdots \langle a_j, x_s\rangle , x_{s+1} \right\rangle = 0.$$

Since the scalar product $\langle \cdot, \cdot \rangle$ is non-degenerate, it means that

$$\Phi(x_1, \dots, x_s) = \sum_{j=1}^{t} \lambda_i \langle a_j, x_1 \rangle \cdots \langle a_j, x_t \rangle a_j.$$

Hence Φ admits a hypersymmetric decomposition. The other assertions follow. □

Definition 4.1.10 (Hypersymmetric map)**.** An s-linear map

$$\Phi : V^s \to V$$

is called *hypersymmetric* if the associated $(s+1)$-linear form $\widetilde{\Phi}$ is symmetric.

The most important case is arguably the bilinear case, where $s = 2$, because it was thoroughly studied. For that reason, we sometimes replace the word hypersymmetric by *trisymmetric* in that particular case, because of the form of the formulas

$$\Phi(x, y) = \sum_{j=1}^{t} \lambda_j \langle a_j, x \rangle \langle a_j, y \rangle a_j$$

that includes the same element a_j three times, and we write $\mu^{\mathrm{tri}}(\Phi)$ instead of $\mu^{\mathrm{hyp}}(\Phi)$. Lemma 4.1.9 states that, if $q \geq 3$, a trisymmetric map Φ always admits a trisymmetric decomposition.

4.1.3 Galois invariance

Another type of interesting decompositions is Galois invariant decompositions, that we also call σ-invariant decompositions. It is motivated by the study of group actions on the set of decompositions, that can sometimes be used to cut branches in the search tree of the algorithms [Cov19]. Let

$$\begin{array}{rccc} \sigma : & V & \to & V \\ & x & \mapsto & x^\sigma \end{array}$$

be a $\mathbf{k}$-automorphism of V that respects the scalar product, *i.e.* for all $x, y \in V$, we have

$$\langle x^\sigma, y^\sigma \rangle = \langle x, y \rangle .$$

Then, if σ is also compatible with some multilinear map Φ, it induces an action on the set of decompositions, as explained in Lemma 4.1.11.

Lemma 4.1.11. *Let V be a finite-dimensional $\mathbf{k}$-vector space and*

$$\Phi : V^s \to V$$

be a symmetric s-linear map that is compatible with σ, i.e. *for all $x_1, \dots, x_s$ in V, we have*

$$\Phi(x_1^\sigma, \dots, x_s^\sigma) = \Phi(x_1, \dots, x_s)^\sigma.$$

If $(a_j)_{1 \leq j \leq t}$ and $(b_j)_{1 \leq j \leq t}$ define a symmetric formula for Φ

$$\Phi(x_1, \dots, x_s) = \sum_{j=1}^{t} \langle a_j, x_1 \rangle \dots \langle a_j, x_s \rangle b_j,$$

then $(a_j^\sigma)_{1 \leq j \leq t}$ and $(b_j^\sigma)_{1 \leq j \leq t})$ also define a symmetric formula for Φ

$$\Phi(x_1, \dots, x_s) = \sum_{j=1}^{t} \langle a_j^\sigma, x_1 \rangle \dots \langle a_j^\sigma, x_s \rangle b_j^\sigma.$$

Proof. Assume that we have a symmetric decomposition for Φ, with the same notations as in the Lemma. First, notice that for every $x, y \in V$, we have

$$\langle x^\sigma, y\rangle = \langle x, y^{\sigma^{-1}}\rangle.$$

Then, it follows that

$$\begin{aligned}\Phi(x_1,\dots,x_s) &= \Phi(x_1^{\sigma^{-1}},\dots,x_s^{\sigma^{-1}})^\sigma \\ &= (\sum_{j=1}^{t}\langle a_j, x_1^{\sigma^{-1}}\rangle \dots \langle a_j, x_s^{\sigma^{-1}}\rangle b_j)^\sigma \\ &= \sum_{j=1}^{t}\left\langle a_j^\sigma, x_1\right\rangle \dots \left\langle a_j^\sigma, x_s\right\rangle b_j^\sigma.\end{aligned}$$

Thus we have a new symmetric formula for Φ. □

When this action does not change the formula, we then say that it is σ-invariant.

Definition 4.1.12 (σ-invariance)**.** Let $(a_j)_{1\le j\le t}$ and $(b_j)_{1\le j\le t}$ define a symmetric formula for Φ

$$\Phi(x_1,\dots,x_s) = \sum_{j=1}^{t}\left\langle a_j, x_1\right\rangle \dots \left\langle a_j, x_s\right\rangle b_j.$$

We say that this formula is σ-invariant if it is the same as the formula defined by $(a_j^\sigma)_{1\le j\le t}$ and $(b_j^\sigma)_{1\le j\le t}$, *i.e.* if there is a permutation $\pi \in \mathfrak{S}_t$ of $\{1,\dots,t\}$ such that $(a_j^\sigma, b_j^\sigma) = (a_{\pi(j)}, b_{\pi(j)})$ for all j. This also applies to hypersymmetric formulas, setting $b_j = \lambda_j a_j$.

Example 4.1.13. In fact, the trisymmetric formula seen in Example 4.1.8 was already σ-invariant. Recall that we work in $\mathbb{F}_4$, defined by

$$\mathbb{F}_4 \cong \mathbb{F}_2[T]/(T^2+T+1) \cong \mathbb{F}_2(\zeta),$$

and we take

$$\mathrm{Tr}(xy) = \langle x, y\rangle.$$

The $\mathbb{F}_2$-automorphism that we consider is the Frobenius automorphism

$$\begin{array}{rccc}\sigma: & \mathbb{F}_4 & \to & \mathbb{F}_4 \\ & x & \mapsto & x^2.\end{array}$$

We still have, for all $x, y \in \mathbb{F}_4$

$$xy = \langle 1, x\rangle\langle 1, y\rangle \cdot 1 + \langle 1+\zeta, x\rangle\langle 1+\zeta, y\rangle \cdot (1+\zeta) + \langle \zeta, x\rangle\langle \zeta, y\rangle \cdot \zeta.$$

Since

$$\begin{cases} 1^\sigma = 1 \\ \zeta^\sigma = \zeta + 1 \\ (\zeta+1)^\sigma = \zeta \end{cases}$$

we see that the formula is σ-invariant.

4.1.4 Multiplication formulas in algebras

In the previous pages, we defined (hyper)symmetric formulas for any multilinear map defined over some finite-dimensional $\mathbf{k}$-vector space. Nevertheless, as seen in the examples, we are often interested in special instances of multilinear maps. In fact, the map that we have in mind is almost always the binary product of some $\mathbf{k}$-algebra $\mathcal{A}$. There are two types of algebras we are particularly interested in.

- The finite field extensions $\mathcal{A} = \mathbb{F}_{q^k}$, in which we take the trace bilinear form

  $$\langle x, y \rangle = \mathrm{Tr}(xy)$$

  for our scalar product, and the Frobenius automorphism

  $$\begin{array}{rccc} \sigma : & \mathbb{F}_{q^k} & \to & \mathbb{F}_{q^k} \\ & x & \mapsto & x^q. \end{array}$$

  for our $\mathbf{k}$-automorphism.

- Algebras of truncated polynomials $\mathcal{A} = \mathbb{F}_q[T]/(T^k)$. In this case, we let

  $$\begin{array}{rccc} \tau : & \mathcal{A} & \to & \mathbf{k} \\ & \sum_{j=0}^{k-1} x_j T^j & \mapsto & x_{k-1} \end{array}$$

  and

  $$\langle x, y \rangle = \tau(xy).$$

  Indeed, if $x = \sum_{j=0}^{k-1} x_j T^j$ and $y = \sum_{j=0}^{k-1} y_j T^j$, we have

  $$\tau(xy) = x_0 y_{k-1} + x_1 y_{k-2} + \cdots + x_{k-1} y_0,$$

  thus $\langle \cdot, \cdot \rangle$ is a non-degenerate bilinear form.

We denote by $\mu_q^{\mathrm{tri}}(k)$ the trisymmetric bilinear complexity of the 2-variable product in $\mathbb{F}_{q^k}$ and by $\hat{\mu}_q^{\mathrm{tri}}(k)$ the trisymmetric bilinear complexity of the 2-variable product in $\mathbb{F}_q[T]/(T^k)$.

4.2 Algorithmic search in small dimension

In very small dimension, *e.g.* in Examples 3.1.3 and 4.1.8 where we are working with $\mathbf{k}$-vector spaces of dimension 2, the search for formulas can be done by hand relatively easily because the length of the formulas is short. Though, even in small dimension, if the size of $\mathbf{k}$ is large, the number of different formulas can be big, thus making it hard to list *all* different solutions. For these reasons, it is highly desirable to *algorithmically* find the formulas. Let V be a finite dimensional $\mathbf{k}$-vector space and $\Phi : V \times V \to V$ a bilinear map. When wanting to list all the symmetric formulas of the form

$$\Phi(x, y) = \sum_{j=0}^{t} \varphi_j(x)\varphi_j(y)a_j,$$

Barbulescu *et al.*'s and Covanov's algorithms exhaustively search through all linear forms φ, cleverly eliminating useless branches in the search tree. Nevertheless, their methods exploit the

fact that it is possible to freely choose the vectors $a_j \in V$, independently of the linear forms $\varphi_j \in V^\vee$. This is no longer possible when searching for trisymmetric decompositions

$$\Phi(x,y) = \sum_{j=0}^{t} \lambda_j \langle a_j, x\rangle \langle a_j, y\rangle a_j,$$

because each linear form $\varphi \in V^\vee$ is linked with a unique vector $a \in V$ such that for all $x \in V$

$$\varphi(x) = \langle a, x\rangle,$$

and the choice of a linear form φ_j with $\varphi_j(x) = \langle a_j, x\rangle$ imposes the choice of the vector $a_j \in V$. We thus propose an *ad hoc* algorithm to find trisymmetric formulas, exploiting the vector space structure.

4.2.1 General algorithm description

In all the section, we assume that $\mathbf{k} = \mathbb{F}_q$ is the finite field with q elements, V is a finite-dimensional $\mathbf{k}$-vector space equipped with a non-degenerate bilinear form, written as a scalar product $\langle\cdot,\cdot\rangle$, and $\Phi : V \times V \to V$ is a hypersymmetric bilinear map for which we want to compute trisymmetric decompositions

$$\Phi(x,y) = \sum_{j=1}^{t} \lambda_j \langle a_j, x\rangle \langle a_j, y\rangle a_j.$$

Assume that

$$V \cong \mathbf{k}^k$$

is a $\mathbf{k}$-vector space of dimension k, for which a basis has been chosen and allows us to identify V to $\mathbf{k}^k$, and let $(b_j)_{1\leq j\leq k}$ be the projections of Φ on each coordinate, *i.e.* for all $x, y \in V$, we have

$$\Phi(x,y) = (b_1(x,y), \ldots, b_k(x,y)).$$

We already saw that the difficulty in the trisymmetric case resides in the fact that each linear form, or equivalently each symmetric rank 1 bilinear form, comes with a given vector in V that dictates its impact on the different coordinates in a trisymmetric formula. Therefore, a central idea is to exhaustively search through vectors in V instead of linear forms in $V^\vee$. This is equivalent since

$$V \cong V^\vee$$

in this case anyway. Moreover, we search through special sets of vectors that have easy-to-manage coordinates, *e.g.* well-placed zeros and ones, in order to control the impact on certain coordinates. Assume $(a_j)_{1\leq j\leq t}$ in V and $(\lambda_j)_{1\leq j\leq t}$ in $\mathbf{k}$ define a trisymmetric decomposition

$$\Phi(x,y) = \sum_{j=1}^{t} \lambda_j \langle a_j, x\rangle \langle a_j, y\rangle a_j.$$

Without loss of generality, we can consider that every element a_j is "normalized", *i.e.* its first nonzero coordinate is 1. Indeed, if we have one

$$a_{j_0} = 0$$

for some $1 \leq j_0 \leq t$, then we just remove one term from the formula and we still have a trisymmetric decomposition, of length $t-1$. Now if for every $1 \leq j \leq t$,

$$a_j \neq 0,$$

we let x_j be the first nonzero coordinate of a_j and we write

$$a_j = x_j \widetilde{a}_j.$$

We can now write the trisymmetric formula as

$$\begin{aligned}
\Phi(x,y) &= \sum_{j=1}^{t} \lambda_j \langle a_j, x\rangle \langle a_j, y\rangle a_j \\
&= \sum_{j=1}^{t} \lambda_j \langle x_j\widetilde{a}_j, x\rangle \langle x_j\widetilde{a}_j, y\rangle x_j\widetilde{a}_j \\
&= \sum_{j=1}^{t} \lambda_j x_j^3 \langle \widetilde{a}_j, x\rangle \langle \widetilde{a}_j, y\rangle \widetilde{a}_j \\
&= \sum_{j=1}^{t} \widetilde{\lambda}_j \langle \widetilde{a}_j, x\rangle \langle \widetilde{a}_j, y\rangle \widetilde{a}_j,
\end{aligned}$$

where $\widetilde{\lambda}_j = \lambda_j x_j^3$. Therefore any trisymmetric formula is equivalent to a trisymmetric formula with normalized vectors, and thus we only search for formulas with normalized elements. In other words, for all $1 \leq i \leq k$, we let

$$\mathcal{E}_i = \{x = (x_1, \ldots, x_k) \in V \,|\, \forall l \leq i-1,\, x_l = 0 \text{ and } x_i = 1\}$$

and

$$\mathcal{E} = \bigcup_{i=1}^{k} \mathcal{E}_i,$$

and we search for elements a_j in $\mathcal{E}$ instead of the entire vector space V. Limiting the search to $\mathcal{E}$ helps us in two different ways. First, it reduces the complexity of the exhaustive search since the size of $\mathcal{E}$ is smaller than the size of V. Indeed, the sets $\mathcal{E}_i$ are disjoint, so

$$\begin{aligned}
\#\mathcal{E} &= \sum_{i=1}^{k} \#\mathcal{E}_i \\
&= \sum_{i=1}^{k} q^{k-i} \\
&= \frac{q^k - 1}{q-1},
\end{aligned}$$

whereas

$$\#V = q^k.$$

Second, it leads to a better understanding of what happens in the algorithm, because if we have some vector

$$a \in \mathcal{E}_i$$

for a given $1 \leq i \leq k$, we know that the associated bilinear form

$$(x, y) \mapsto \langle a, x \rangle \langle a, y \rangle$$

can only impact the coordinates $l \geq i$. Thus, we further use the vector space structure of V by searching for solutions on each coordinate, starting with the first coordinates and vectors in $\mathcal{E}_1$, then the second coordinate and vectors in $\mathcal{E}_2$, and so on until the last coordinate. Let us focus on the first coordinate and give some details.

Recall that the goal is to obtain a trisymmetric decomposition for the hypersymmetric bilinear map $\Phi : V \times V \to V$, that is written

$$\Phi(x, y) = (b_1(x, y), \ldots, b_k(x, y)).$$

in the basis of V. We first see how to decompose the bilinear *form* b_1 as a sum of rank 1 bilinear forms. Let $\mathcal{B}$ be the set of bilinear forms of $V \times V$, recall that $\mathcal{B}$ is a $\mathbf{k}$-vector space of dimension k^2, and that we identify b_1 with the $k \times k$ matrix $B_1 \in \mathbf{k}^{k \times k}$ such that for all vectors $x, y \in V$, we have

$$b_1(x, y) = X B_1 Y^t,$$

where $X, Y \in \mathbf{k}^{1 \times k}$ are the row vectors representing x and y and where Y^t is the transpose of y. Let r_1 be the rank of b_1, we know that b_1 can be decomposed as a sum of r_1 bilinear forms of rank 1. Let f be the application mapping an element in V to its associated bilinear form:

$$\begin{array}{rccc} f: & V & \to & \mathcal{B} \\ & a & \mapsto & (x, y) \mapsto \langle a, x \rangle \langle a, y \rangle . \end{array}$$

In order to find these decompositions, we begin by exhaustively searching through scalars $\lambda_1 \in \mathbf{k}$ and vectors $a_1 \in \mathcal{E}_1$ such that

$$r_1 - 1 = \operatorname{rank}(b_1 - \lambda_1 f(a_1)) < \operatorname{rank}(b_1) = r_1.$$

Then, for each such pair (λ_1, a_1), we exhaustively search through scalars $\lambda_2 \in \mathbf{k}$ and vectors $a_2 \in \mathcal{E}_1$ such that

$$r_1 - 2 = \operatorname{rank}(b_1 - \lambda_1 f(a_1) - \lambda_2 f(a_2)) < \operatorname{rank}(b_1 - \lambda_1 f(a_1)) = r_1 - 1.$$

We continue this process until we have r_1 pairs $(\lambda_1, a_1), \ldots, (\lambda_{r_1}, a_{r_1})$ such that

$$0 = \operatorname{rank}(b_1 - \sum_{j=1}^{r_1} \lambda_j f(a_j)) < \operatorname{rank}(\sum_{j=1}^{r_1 - 1} \lambda_j f(a_j)) = 1,$$

which exactly means that

$$b_1 = \sum_{j=1}^{r_1} \lambda_j f(a_j),$$

and we have found our decomposition. In fact, we can search in a more clever way. Since the rank of b_1 is r_1 and we are looking for decompositions as a sum of exactly r_1 bilinear forms of rank 1, we must choose pairs (λ_j, a_j) that decrease the rank of our bilinear form at each step, otherwise we will need strictly more than r_1 bilinear forms of rank 1. Therefore, at each step, we can search

only through the vectors that can decrease the rank of the last considered bilinear form. After each choice of (λ_1, a_1), we will thus exhaustively search through scalars $\lambda_2 \in \mathbf{k}$ and vectors a_2 in

$$\mathcal{E}_1^{\{(\lambda_1,a_1)\}} = \{a \in \mathcal{E}_1 \mid \exists \lambda \in \mathbf{k},\, \mathrm{rank}(b_1 - \lambda f(a)) < \mathrm{rank}(b_1)\}\,.$$

Then, after each choice of (λ_2, a_2), we will search through scalars $\lambda_3 \in \mathbf{k}$ and vectors a_3 in

$$\mathcal{E}_1^{\{(\lambda_1,a_1),(\lambda_2,a_2)\}} = \left\{a \in \mathcal{E}_1^{\{(\lambda_1,a_1)\}} \mid \exists \lambda \in \mathbf{k},\, \mathrm{rank}(b_1 - \lambda_1 f(a_1) - \lambda f(a)) < \mathrm{rank}(b_1 - \lambda_1 f(a_1))\right\},$$

and we use the same idea until the end of the process. This strategy, used in Algorithm 3, saves a lot of time in the search because we look only at the vectors that have the potential to decrease the rank, instead of all the vectors in $\mathcal{E}_1$.

Algorithm 3 (Minimal decomposition)

Input: $b \in \mathcal{B}$ a bilinear form of rank r, $E \subset \mathcal{E}$ the space of vectors where to search

Output: A list of decompositions of b as a sum of bilinear forms of rank 1, each decomposition represented by a set of r pairs $\{(\lambda_1, a_1), \ldots, (\lambda_r, a_r)\}$.

procedure MinimalDecomposition($b, E, R_{\mathrm{glob}}, R_{\mathrm{loc}} = \emptyset$)
  if $\varphi = 0$ **then** ▷ $\mathrm{rank}(\varphi) = 0$
    $R_{\mathrm{glob}} \leftarrow R_{\mathrm{glob}} \bigcup \{R_{\mathrm{loc}}\}$
  else
    $\mathcal{C} \leftarrow \emptyset$ ▷ $\mathcal{C}$ is the set of pairs that decrease the rank
    for all $a \in E$ **do**
      for all $\lambda \in \mathbf{k}$ **do**
        if $\mathrm{rank}(b - \lambda f(a)) < \mathrm{rank}(b)$ **then**
          $\mathcal{C} \leftarrow \mathcal{C} \bigcup \{(\lambda, a)\}$
          break ▷ breaks only the inner loop
        end if
      end for
    end for
    for $i = 1$ to $\#\mathcal{C}$ **do** ▷ we note $\mathcal{C} = \{(\gamma_1, c_1), \ldots, (\gamma_u, c_u)\}$
      $E' \leftarrow \{c_{i+1}, \ldots, c_u\}$
      MinimalDecomposition($b - \gamma_i f(c_i), E', R_{\mathrm{glob}}, R_{\mathrm{loc}} \cup \{(\gamma_i, c_i)\}$)
    end for
  end if
end procedure
$\mathcal{R} \leftarrow \emptyset$
MinimalDecomposition($b, E, \mathcal{R}$)
return $\mathcal{R}$

For each decomposition of b_1

$$b_1(x, y) = \sum_{j=1}^{r_1} \lambda_j \langle a_j, x\rangle \langle a_j, y\rangle$$

that we compute this way, we obtain

$$\Phi(x, y) - \sum_{j=1}^{r_1} \lambda_j \langle a_j, x\rangle \langle a_j, y\rangle a_j = (0, b'_2(x, y), \ldots, b'_k(x, y)),$$

where $b'_2, \ldots, b'_k \in \mathcal{B}$ are new bilinear forms, depending on the coordinates of $a_1, \ldots, a_{r_1}$, and where the first coordinate is 0 because the first coordinate of the vectors $a_1, \ldots, a_{r_1}$ is always 1. Then, we use Algorithm 3 with the bilinear form b'_2 of rank r_2 and the initial set of vectors $\mathcal{E}_2$. For each decomposition of b'_2

$$b'_2(x,y) = \sum_{j=r_1+1}^{r_1+r_2} \lambda_j \langle a_j, x\rangle \langle a_j, y\rangle$$

obtained, we have

$$\Phi(x,y) - \sum_{j=1}^{r_1+r_2} \lambda_j \langle a_j, x\rangle \langle a_j, y\rangle a_j = (0, 0, b''_3(x,y), \ldots, b''_k(x,y)),$$

where $b''_3, \ldots, b''_k \in \mathcal{B}$ are again new bilinear forms, depending on the coordinates of the elements $a_{r_1+1}, \ldots, a_{r_1+r_2}$. We continue this process on all the coordinates, such that in the end we obtain trisymmetric decompositions of the form

$$\Phi(x,y) = \sum_{j=1}^{t} \lambda_j \langle a_j, x\rangle \langle a_j, y\rangle a_j.$$

The overall strategy is described in Algorithm 4.

Algorithm 4 (Trisymmetric search with minimal decompositions)

Input: $\Phi = (b_1, \ldots, b_k)$ a bilinear map; $t \in \mathbb{N}$ an integer
Output: A list of trisymmetric decompositions of Φ of length up to t.

procedure TriSymSearchMin($\Phi, t, S_{\text{glob}}, i = 1, S_{\text{loc}} = \emptyset$)
  if $\Phi = 0$ **then**
    $S_{\text{glob}} \leftarrow S_{\text{glob}} \bigcup \{S_{\text{loc}}\}$.
  else if $\text{rank}(b_i) \leq t$ **then**
    $\mathcal{R} \leftarrow \emptyset$
    MinimalDecomposition($b_i, \mathcal{E}_i, \mathcal{R}$) ▷ We have $\Phi = (b_1, \ldots, b_k)$
    for all $S \in \mathcal{R}$ **do**
      $\Phi' \leftarrow \Phi - \sum_{(\lambda,\alpha)\in S} \lambda f(\alpha)\alpha$
      TriSymSearchMin($\Phi', t - \#S, S_{\text{glob}}, i+1, S_{\text{loc}} \cup S$)
    end for
  end if
end procedure
$\mathcal{S} \leftarrow \emptyset$
TriSymSearchMin($\Phi, t, \mathcal{S}$)
return $\mathcal{S}$

Let $(\lambda_j)_{1\leq j\leq t}$ be some scalars in $\mathbf{k}$ and $(a_j)_{1\leq j\leq t}$ some vectors in

$$\mathcal{E} = \bigcup_{i=1}^{k} \mathcal{E}_i$$

that define an optimal trisymmetric decomposition for Φ (*i.e.* the length of the decomposition is minimal):

$$\Phi(x,y) = \sum_{j=1}^{t} \lambda_j \langle a_j, x\rangle \langle a_j, y\rangle a_j.$$

Although the formula is optimal, it is entirely possible for the individual coordinate decompositions to be sub-optimal. Assume that for all vectors $x, y \in V$, we have

$$\Phi(x, y) = (b_1(x, y), \dots, b_k(x, y)),$$

with r_1 the rank of the bilinear form $b_1 \in \mathcal{B}$. If we have strictly more than r_1 different vectors in $(a_j)_{1 \leq j \leq t}$ that belong to $\mathcal{E}_1$, then Algorithm 4 will not find this decomposition. Indeed, Algorithm 3 only finds *minimal* decompositions of b_1 into r_1 bilinear forms of rank 1, *i.e.* finds decompositions with exactly r_1 vectors in $(a_j)_{1 \leq j \leq t}$ that are in $\mathcal{E}_1$. More generally, in Algorithm 3, when working with a bilinear form b of rank r, it is possible that the best *local* decompositions (*i.e.* on only one coordinate) of length r are not the best in order to find *global* decompositions (*i.e.* on all the coordinates), because of the impact the decompositions of b have on the other coordinates. For that reason, it is important to add the option in Algorithm 3 to search for non-optimal decompositions. That is exactly what is done in Algorithm 5: if we want to find all decompositions of some bilinear form b of rank r of length $r + \mu$, with $\mu > 0$, we still exhaustively search through scalars $\lambda \in \mathbf{k}$ and vectors $a \in \mathcal{E}$, but we allow the rank *not to* decrease on μ different times. Once this number of "exceptions" have all been used, we go back to the strategy previously presented (*i.e.* Algorithm 3).

We let μ_j be the number of times we allow the rank not to decrease when dealing with the j-th coordinate during the trisymmetric search, and we let

$$\mathfrak{M} = (\mu_1, \dots, \mu_k).$$

We call this m-uple the *margin* of the exhaustive search. Algorithm 6 is then a generalization of Algorithm 4 that includes the notion of margin. In the other direction, Algorithm 4 is the special case of Algorithm 6 with the margin

$$\mathfrak{M} = (0, \dots, 0).$$

Algorithm 6 behaves differently, both in performance and in number of decompositions found, when used with a variety of margins. Example 4.2.1 shows a simple case of this phenomenon on a small field extension.

Example 4.2.1. In order to illustrate the impact of the choice of a given margin, let us see the case of the extension

$$\mathbb{F}_{3^3} \cong \mathbb{F}_3[x]/(x^3 - x + 1) \cong \mathbb{F}_3[z]$$

over $\mathbb{F}_3$. There are 4 trisymmetric decompositions of length 6 of the product in $\mathbb{F}_{3^4}$:

$$\begin{aligned}
d_1 &= \left\{(1, z^2 + z), (2, z^2 + z + 1), (1, z^2 - z + 1), (1, -z^2 + 1), (2, -z^2 + z + 1), (1, -z^2 - z + 1)\right\} \\
d_2 &= \left\{(2, z^2), (2, z^2 + 1), (1, z^2 + z + 1), (2, -z^2 + 1), (1, -z^2 + z), (2, -z^2 - z + 1)\right\} \\
d_3 &= \left\{(1, z), (1, z + 1), (2, -z + 1), (2, z^2 + z), (1, -z^2 + 1), (1, -z^2 + z)\right\} \\
d_4 &= \left\{(1, z^2), (1, z^2 + 1), (2, z^2 + z), (2, z^2 - z + 1), (2, -z^2 + z), (1, -z^2 + z + 1).\right\}
\end{aligned}$$

How the elements of each decomposition fall into the sets $\mathcal{E}_1$, $\mathcal{E}_2$ or $\mathcal{E}_3$ is described in Table 4.1. Table 4.2 describes which choice of margin produces which decompositions. There is a checkmark ✓when the algorithm succeeds in finding the decomposition.

Algorithm 5 (Decomposition with margin)

Input: $b \in \mathcal{B}$ a bilinear form of rank r, $E \subset \mathcal{E}$ the space of vectors in which to search, μ the margin, t the maximum length of a decompositon of b

Output: A list of decompositions of b as a sum of at bilinear forms of rank 1, each decomposition represented by a set of at most t pairs $\{(\lambda_1, a_1), \ldots, (\lambda_t, a_t)\}$.

procedure DecompositionWithMargin($b, E, r, \mu, R_{\text{glob}}, R_{\text{loc}} = \emptyset$)
  if $\operatorname{rank} b \leq t$ **then**
    if $b = 0$ **then** ▷ $\operatorname{rank} b = 0$
      $R_{\text{glob}} \leftarrow R_{\text{glob}} \bigcup \{R_{\text{loc}}\}$
    else if $\mu = 0$ **then** ▷ No margin left!
      MinimalDecomposition($\varphi, E, R_{\text{glob}}, R_{\text{loc}}$) ▷ "naive" strategy: Alg. 3
    else
      for $i = 1$ to $\#E$ **do** ▷ We note $E = \{c_1, \ldots, c_u\}$
        $E' \leftarrow \{c_{i+1}, \ldots, c_u\}$
        for all $\lambda \in \mathbb{F}_p$ **do**
          $b' \leftarrow b - \lambda f(c_i)$
          $R'_{\text{loc}} \leftarrow R_{\text{loc}} \bigcup \{(\lambda, c_i)\}$
          if $\operatorname{rank}(b') < \operatorname{rank}(b)$ **then**
            DecompositionWithMargin($b', E', t-1, \mu, R_{\text{glob}}, R'_{\text{loc}}$)
          else
            DecompositionWithMargin($b', E', t-1, \mu-1, R_{\text{glob}}, R'_{\text{loc}}$)
          end if
        end for
      end for
    end if
  end if
end procedure
$\mathcal{R} \leftarrow \emptyset$
DecompositionWithMargin($b, E, \mu, \mathcal{S}$)
return $\mathcal{R}$

Decomposition \ Set	$\mathcal{E}_1$	$\mathcal{E}_2$	$\mathcal{E}_3$
d_1	5	1	0
d_2	4	1	1
d_3	3	3	0
d_4	3	2	1

Table 4.1: The decompositions of the product in $\mathbb{F}_{3^3}$ and how many of their vectors are in each set $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}_3$, see Example 4.2.1 for details.

Algorithm 6 (Trisymmetric search with margins)

Input: $\Phi = (b_1, \ldots, b_k)$ a bilinear map; $t \in \mathbb{N}$ an integer, $\mathfrak{M}$ a margin
Output: A list of trisymmetric decompositions of Φ with up to t elements.

procedure TriSymSearchMargin($\Phi, t, \mathfrak{M}, S_{\text{glob}}, i = 1, S_{\text{loc}} = \emptyset$)
  if $\Phi = 0$ **then**
    $S_{\text{glob}} \leftarrow S_{\text{glob}} \bigcup \{S_{\text{loc}}\}$.
  else if $\text{rank}(b_i) \leq t$ **then**
    $\mathcal{R} \leftarrow \emptyset$ ▷ We note $\mathfrak{M} = (\mu_1, \ldots, \mu_k)$
    DecompositionWithMargin($b_i, \mathcal{E}_i, t, \mu_i, \mathcal{R}$) ▷ We note $\Phi = (b_1, \ldots, b_k)$
    for all $S \in \mathcal{R}$ **do**
      $\Phi' \leftarrow \Phi - \sum_{(\lambda,\alpha)\in S} \lambda f(\alpha)\alpha$
      TriSymSearchMargin($\Phi', t - \#S, \mathfrak{M}, S_{\text{glob}}, i + 1, S_{\text{loc}} \cup S$)
    end for
  end if
end procedure
$\mathcal{S} \leftarrow \emptyset$
TriSymSearchMargin($\Phi, t, \mathfrak{M}, \mathcal{S}$)
return $\mathcal{S}$

Decomposition \ Margin	$(0,0,0)$	$(1,0,0)$	$(0,1,0)$	$(1,1,0)$	$(2,1,0)$
d_1					✓
d_2		✓		✓	✓
d_3			✓	✓	✓
d_4	✓	✓	✓	✓	✓

Table 4.2: The decompositions of the product in $\mathbb{F}_{3^3}$ and whether our algorithm finds them, depending on the margin used. See Example 4.2.1 for details.

4.2.2 Implementation

The ideas discussed in Section 4.2.1, in particular Algorithms 5 and 6, were implemented using Nemo [H+16], a number theory library written in the Julia programming language [Jul]. Nemo functionalities includes a wrapper for Flint [Har10], a number theory library written in the C programming language. The code is available online at `https://github.com/erou/TriSym.jl`.

Julia and Nemo. Julia is a free and open-source, high-level programming language developed since 2012, with dynamic type system and high-performance. It is a compiled language, with a just-in-time (jit) compilation, meaning that the compilation is almost invisible for the user but performances are comparable with other compiled languages. It is easy to learn and to read, especially for someone who already worked with a high-level language like Python. There are of course many other things to say about this new language, the interested reader can find additional information on Julia's website[1] and, of course, in the documentation of the language. Julia is designed for numerical computing, but there are many other possibilities, *e.g.* some symbolic computer algebra packages, such as Nemo. Nemo is a computer algebra package for Julia. It aims to cover commutative algebra, number theory and group theory. It contains wrappers for MPIR, Flint, Arb and Antic, and other features written directly in Julia. Among these libraries, the C library Flint (Fast library for number theory) is the only one that we use. One of the advantages of Julia is that it is fairly easy to directly call C functions contained in other libraries. Therefore, all the Nemo functions that we use in TriSym.jl are in fact Flint functions. This provides efficiency, the main reason why we use the duo Julia/Nemo, as well as simplicity, because we are not directly using the C programming language. On top of that, if a Flint function is not available in Nemo, we can directly call it from Julia. Moreover, it is also possible to write crucial parts of the code directly in C and call the code if necessary.

Trisymmetric search in finite fields. An algorithm in our Julia library `Trisym.jl` is specially designed to handle trisymmetric decompositions of the product in finite fields. We use a lot of native Nemo types, that are wrappers for Flint types. Finite field elements are represented by univariate polynomials modulo an irreducible polynomial and only prime field extensions

$$\mathbb{F}_{p^k}/\mathbb{F}_p,$$

where p is a prime number, can be considered. Two types represent finite fields in Flint: `fq` and `fq_nmod`, the latter only works for word-size primes p. Because the algorithms for trisymmetric searches have exponential complexity, we focused our implementation on finite fields with a word-size characteristic, but there are no theoretical nor implementation obstacles preventing an implementation with the type `fq` and arbitrary large characteristic. Nonetheless, implementing the algorithms for any kind of extension

$$\mathbb{F}_{q^k}/\mathbb{F}_q,$$

where q is a prime power, would require additional work. The bilinear forms in $\mathbb{F}_{p^k}$ are represented by $k \times k$ matrices over $\mathbb{F}_p$, and the product in $\mathbb{F}_{p^k}$ is represented by a k-uple of $k \times k$ matrices. The matrix manipulations are all done by Flint using efficient C code: it is in particular true for the rank computation, that plays a central role in Algorithm 5. We always precompute a dictionary mapping elements in the finite field

$$a \in \mathbb{F}_{p^k}$$

[1] `https://julialang.org/`

Field	Margin	Solutions	Length	Time (s)
$\mathbb{F}_{3^2}$	$(0,0)$	1	3	$8.5 \cdot 10^{-5}$
$\mathbb{F}_{3^3}$	$(0,0,0)$	1	6	$4.0 \cdot 10^{-4}$
$\mathbb{F}_{3^3}$	$(2,1,0)$	4	6	$6.2 \cdot 10^{-3}$
$\mathbb{F}_{3^4}$	$(0,0,0,0)$	2	9	$1.6 \cdot 10^{-3}$
$\mathbb{F}_{3^4}$	$(1,0,0,0)$	5	9	$2.5 \cdot 10^{-2}$
$\mathbb{F}_{3^4}$	$(0,1,0,0)$	5	9	$3.9 \cdot 10^{-3}$
$\mathbb{F}_{3^4}$	$(1,1,0,0)$	10	9	$2.7 \cdot 10^{-2}$
$\mathbb{F}_{3^4}$	$(2,1,0,0)$	18	9	$1.6 \cdot 10^{-1}$
$\mathbb{F}_{3^4}$	$(2,1,2,2)$	18	9	$1.6 \cdot 10^{-1}$
$\mathbb{F}_{3^4}$	$(3,2,2,2)$	25	9	$9.1 \cdot 10^{-1}$
$\mathbb{F}_{3^4}$	$(4,3,3,3)$	27	9	4.7
$\mathbb{F}_{3^4}$	$(5,5,5,5)$	27	9	18
$\mathbb{F}_{3^5}$	$(0,0,0,0,0)$	0	11	$7.3 \cdot 10^{-2}$
$\mathbb{F}_{3^5}$	$(1,0,0,0,0)$	0	11	1.6
$\mathbb{F}_{3^5}$	$(1,1,0,0,0)$	0	11	15
$\mathbb{F}_{3^5}$	$(2,1,1,0,0)$	0	11	77
$\mathbb{F}_{3^5}$	$(2,2,1,1,1)$	0	11	100
$\mathbb{F}_{3^5}$	$(3,2,2,2,2)$	0	11	500
$\mathbb{F}_{3^5}$	$(4,2,2,2,2)$	0	11	6100

Table 4.3: Experimental results for $q = 3$.

to the matrix representing the bilinear form

$$(x, y) \mapsto \langle a, x\rangle \langle a, y\rangle .$$

This bilinear form is not hard to compute: it suffices to compute a matrix-vector multiplication, but since we have to compute the same bilinear forms several times, we prefer to store them in a dictionary.

Experimental results All the tests in this section were performed on an Intel Core i7-7500U CPU clocked at 2.70GHz, using Nemo 0.19.1 running on Julia 1.5.3, and Nemo's corresponding version of Flint. The benchmark functions are available in the file `bencharks.jl` of the repository of the package `TriSym.jl`[2], while the data generated from the benchmarks can be found in the repository of the thesis[3]. All plots were made using gnuplot version 5.2 patchlevel 8, and the gnuplot files can also be found in the thesis repository. As previously said, our search algorithm has exponential complexity in the dimension of the algebra we are searching in. This is similar to the algorithms searching for (symmetric) bilinear formulas, such as [Cov19, BDEZ12]. For the smallest case of $q = 3$, we are able to compute solutions up to $k = 4$ with Algorithm 6. For $k = 5$, we know there exist bilinear formulas of length 11, but there are no trisymmetric decomposition of that length that can be found with small margin, thus Algorithm 6 is not able to find a trisymmetric decomposition in dimension $k = 5$. The results are reported in Table 4.3. It is clear that the dimension has an important impact on the timings, but the margin is even more impactful, as can be seen with the different timings obtain for $\mathbb{F}_{3^4}$ and $\mathbb{F}_{3^5}$. As the dimension

[2] https://github.com/erou/TriSym.jl
[3] https://github.com/erou/thesis/

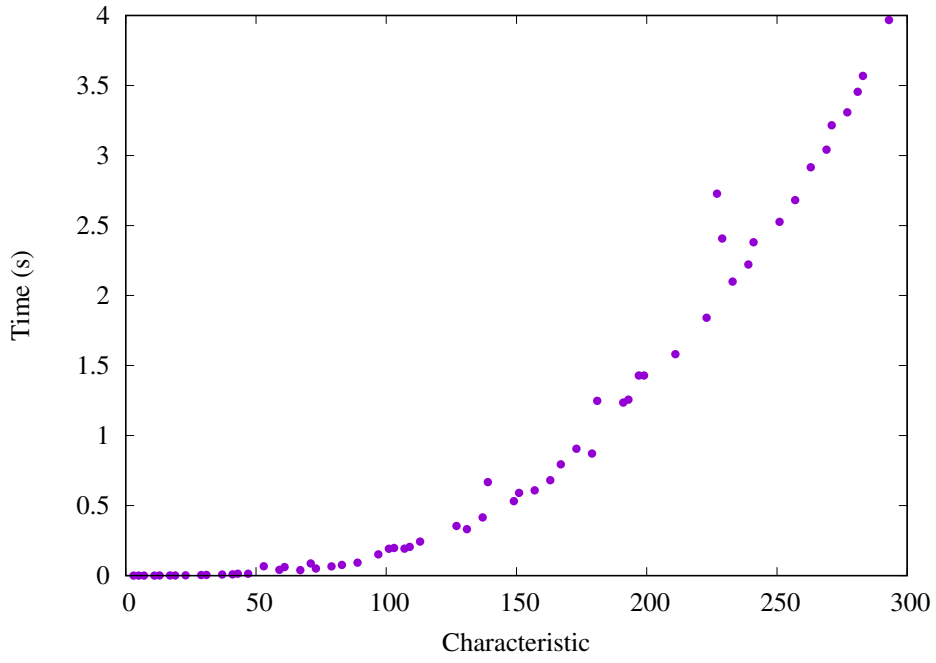


Figure 4.1: Timings for the computation of the trisymmetric decompositions in $\mathbb{F}_{p^2}$ with a margin $\mathfrak{M} = (0, 0)$.

grows, it becomes less and less likely that a formula can be found using a small margin, it thus makes the algorithm relevant mostly in small dimension. The characteristic p of the finite field also has a big influence, because it directly impacts the size of the finite field, and thus the search is longer. We can see in Figure 4.1 the timings related to the computation of all the trisymmetric decompositions in $\mathbb{F}_{p^2}$ for $3 \leq p \leq 300$ with a fixed margin $\mathfrak{M} = (0, 0)$. In these timings, we assume that the dictionary mapping an element $a \in \mathbb{F}_{p^2}$ to its symmetric bilinear form

$$(x, y) \mapsto \langle a, x\rangle \langle a, y\rangle$$

is already precomputed. In practice, this precomputation is about 5 to 10 times faster than the computation of the trisymmetric formulas. In dimension $k = 3$, the exhaustive search is already quite difficult, as there are many formulas. For example, in $\mathbb{F}_{37^3}$, we find 3558 different formulas in about 81 seconds. For that reason, we also plot a version of Algorithm 6 that stops after the discovery of only one formula. This allows us to go further: in Figure 4.2, we show the timings for the computations in $\mathbb{F}_{p^3}$ for $3 \leq p \leq 150$ with a margin $\mathfrak{M} = (0, 0, 0)$. To compare with the last algorithm, the computation in $\mathbb{F}_{37^3}$ now takes 0.13 second. With our algorithms, we are able to compute $\mu_3^{\mathrm{tri}}(3) = 6$, $\mu_p^{\mathrm{tri}}(3) = 5$ for all primes $5 \leq p \leq 257$, $\mu_3^{\mathrm{tri}}(4) = 9$, $\mu_5^{\mathrm{tri}}(4) = 8$, and $\mu_p^{\mathrm{tri}}(4) = 7$ for all primes $7 \leq p \leq 23$. In dimension $k = 5$, these algorithms are not able to find formulas. We also implemented a naive search algorithm for Galois invariant formulas, that exhaustively search through all orbits. Although naive, this algorithm is quite fast for small dimension because the search space is smaller than the one at hand with Algorithm 6. Consequently, it allows us to find Galois invariant formulas of length 11 for $\mathbb{F}_{3^5}$ and of length 10 for $\mathbb{F}_{5^5}$ and $\mathbb{F}_{7^5}$. Joint with the obvious inequalities $\mu_q(k) \leq \mu_q^{\mathrm{sym}}(k) \leq \mu_q^{\mathrm{tri}}(k) \leq \mu_q^{\mathrm{tri},G}(k)$ and with known lower bounds from [BPR+21, Thm. 2.2] and [BDEZ12], this gives

$$10 \leq \mu_3(5) \leq \mu_3^{\mathrm{sym}}(5) = \mu_3^{\mathrm{tri}}(5) = \mu_3^{\mathrm{tri},G}(5) = 11,$$

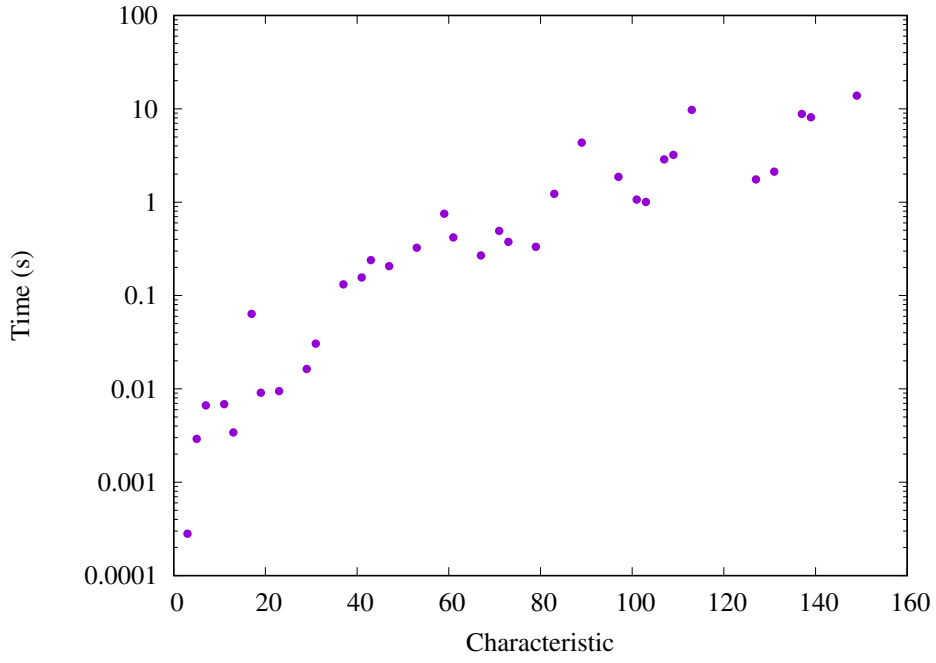


Figure 4.2: Timings for the computation of one trisymmetric decomposition in $\mathbb{F}_{p^3}$ with a margin $\mathfrak{M} = (0, 0, 0)$. Logarithmic scale was used on the y-axis, otherwise only the last points were visible.

$$\mu_5(5) = \mu_5^{\mathrm{sym}}(5) = \mu_5^{\mathrm{tri}}(5) = \mu_5^{\mathrm{tri},G}(5) = 10,$$

and

$$\mu_7(5) = \mu_7^{\mathrm{sym}}(5) = \mu_7^{\mathrm{tri}}(5) = \mu_7^{\mathrm{tri},G}(5) = 10.$$

For $q \geq 3$ we know no example where one of the inequalities in $\mu_q(k) \leq \mu_q^{\mathrm{sym}}(k) \leq \mu_q^{\mathrm{tri}}(k)$ is strict. However, it turns out that the inequality with $\mu_q^{\mathrm{tri},G}(k)$ can be strict. Indeed, let $q = 3$ and $k = 7$. In this setting our exhaustive search found no Galois invariant decomposition of length up to 15. Since all orbits are of length 7, except the trivial orbit of length 1, the minimal length for a Galois invariant decomposition is congruent to 0 or 1 modulo 7, so we deduce that it is at least 21. Furthermore, we know [BPR+21, Table 2] that $\mu_3^{\mathrm{sym}}(7) \leq 19$, so we have

$$\mu_3(7) \leq \mu_3^{\mathrm{sym}}(7) \leq 19 < 21 \leq \mu_3^{\mathrm{tri},G}(7).$$

Although these algorithms are very useful to understand and find trisymmetric bilinear formulas, their algorithmic complexities are prohibitive. With more optimization, one could hope to push a little further the computations, but it is likely that new methods have to be found to really make a breakthrough.

4.2.3 Universal formulas

In our experiments, we were not able to find any example of an algebra

$$\mathcal{A} = \mathbb{F}_{q^k} \text{ or } \mathcal{A} = \mathbb{F}_q[T]/(T^k)$$

where the bilinear complexity and the trisymmetric bilinear complexity are different. In fact, this is an open problem: we do not know whether there exist $q \geq 3$ and $k \geq 2$ such that

$$\mu_q^{\text{tri}}(k) \neq \mu_q(k) \text{ or } \hat{\mu}_q^{\text{tri}}(k) \neq \hat{\mu}_q(k).$$

In fact, for small values of k, we are even able to prove that these quantities are equal, by exhibiting *universal formulas*, *i.e.* formulas that are true for (almost) any choice of $q \geq 3$. In order to obtain such formulas, we slightly change our point of view on the problem. Assume that we want to compute a trisymmetric decomposition of the product

$$\begin{array}{rccc} \Phi: & \mathcal{A} \times A & \to & \mathcal{A} \\ & (x, y) & \mapsto & xy. \end{array}$$

in $\mathcal{A}$, a commutative algebra of degree k over $\mathbb{F}_q$. After the choice of a basis of $\mathcal{A}$ and a basis of the space $\mathcal{B}$ of the bilinear forms on $\mathcal{A}$, we can represent Φ by its coordinates $(\pi_j)_{1 \leq j \leq k}$ in the basis of $\mathcal{B}$. We then see

$$\Phi = (\pi_1, \ldots, \pi_k)$$

as a column vector B of length k^3. The first coordinates correspond to π_1, the next k^2 coordinates correspond to π_2, and so on up to π_k. Now, recall that we let

$$\begin{array}{rccc} f: & \mathcal{A} & \to & \mathcal{B} \\ & a & \mapsto & (x, y) \mapsto \langle a, x \rangle \langle a, y \rangle . \end{array}$$

be the application mapping an element in $\mathcal{A}$ to its associated symmetric bilinear map. We also recall that we let

$$\mathcal{E}_i = \{x = (x_1, \ldots, x_k) \in \mathcal{A} \mid \forall l \leq i - 1,\ x_l = 0 \text{ and } x_i = 1\}$$

and

$$\mathcal{E} = \bigcup_{i=1}^{k} \mathcal{E}_i$$

be subsets of $\mathcal{A}$. Now, for each $a \in \mathcal{E}$, we write

$$\mathbf{f}(a) = a \otimes f(a),$$

where a is the column vector of length k corresponding to a in the basis of $\mathcal{A}$, $f(a)$ is the column vector of length k^2 corresponding to $f(a) \in \mathcal{B}$, and $\otimes$ is the Kronecker product. With these notations, finding a trisymmetric decomposition of the product in $\mathcal{A}$ is the same as finding elements $a_1 \ldots, a_n \in \mathcal{E}$ and $\lambda_1, \ldots, \lambda_n \in \mathbb{F}_q$ with

$$B = \sum_{j=1}^{n} \lambda_j \mathbf{f}(a_j).$$

Let A be the matrix which columns are the $\mathbf{f}(a)$ for all $a \in \mathcal{E}$, then the problem is to find a solution X of

$$AX = B$$

with the smallest possible number of nonzero entries in X. With this new point of view in mind, we first consider the case $\mathcal{A} = \mathbb{F}_{q^2}$ over $\mathbf{k} = \mathbb{F}_q$, where the characteristic of $\mathbf{k}$ is not 2.

Proposition 4.2.2. *For any odd q we have*

$$\mu_q(2) = \mu_q^{tri}(2) = 3.$$

Proof. The fact that $\mu_q(2) = 3$ is not new and follows, for example, from [BPR+21, Thm. 2.2]. In order to prove that $\mu_q^{\text{tri}}(2) = 3$, we find a *universal* trisymmetric formula of length 3. We know that we can find a non-square element ζ in $\mathbb{F}_q$, we can then define

$$\mathbb{F}_{q^2} \cong \mathbb{F}_q[T]/(T^2 - \zeta) = \mathbb{F}_q(\alpha),$$

where $\alpha = \bar{T}$ is the canonical generator of $\mathbb{F}_{q^2}$. Let $x = x_0 + x_1\alpha$ and $y = y_0 + y_1\alpha$ be two elements of $\mathbb{F}_{q^2}$, we have

$$xy = (x_0 + x_1\alpha)(y_0 + y_1\alpha) = x_0y_0 + \zeta x_1y_1 + (x_0y_1 + x_1y_0)\alpha.$$

We can lift the matrix B coming from the multiplication formula, that has coefficients in $\mathbb{F}_q$, to a matrix with coefficients in $\mathbb{Q}(\zeta)$, where ζ is an indeterminate. We can also lift the matrix A, because the map f (and therefore $\mathbf{f}$) has the same expression for all q not divisible by 2. Indeed, one can check that the map f is given by

$$f(x_0 + x_1\alpha) = \left(S\begin{bmatrix}x_0\\x_1\end{bmatrix}\right)\left(S\begin{bmatrix}x_0\\x_1\end{bmatrix}\right)^{\mathsf{T}} = 4\begin{bmatrix}x_0^2 & \zeta x_0x_1\\ \zeta x_0x_1 & \zeta^2x_1^2\end{bmatrix}.$$

where

$$S = \left[\langle \alpha^i, \alpha^j\rangle\right]_{0\le i,j\le 1} = \left[\mathrm{Tr}(\alpha^{i+j})\right]_{0\le i,j\le 1} = \begin{bmatrix}2 & 0\\ 0 & 2\zeta\end{bmatrix}.$$

We can then solve $AX = B$ over $\mathbb{Q}(\zeta)$ and finally check that

$$B = (1-\zeta^{-1})4^{-1}\mathbf{f}(1) + (8\zeta)^{-1}\mathbf{f}(1+\alpha) + (8\zeta)^{-1}\mathbf{f}(1-\alpha),$$

so that the trisymmetric bilinear complexity of $\mathbb{F}_{q^2}/\mathbb{F}_q$ is 3. □

Using the same strategy, we can also find universal formulas for another type of algebra:

$$\mathcal{A} = \mathbb{F}_q[T]/(T^k),$$

namely the truncated polynomials. In that context, we first observe that we have

$$\hat{\mu}_q^{\text{tri}}(k) \ge \hat{\mu}_q(k) \ge 2k - 1$$

for all q and k. Indeed this is a special case of [Win77, Thm. 4], which holds for any polynomial that is a power of an irreducible polynomial. Conversely we are able to find formulas for $2 \le k \le 4$ that match this lower bound.

Proposition 4.2.3. *For any odd q we have*

$$\hat{\mu}_q^{tri}(2) = 3.$$

Proof. Let $\mathcal{A} = \mathbb{F}_q[T]/(T^2) = \mathbb{F}_q[\alpha]$ with $\alpha = \bar{T}$, so $\alpha^2 = 0$. If $x = x_0 + x_1\alpha$ and $y = y_0 + y_1\alpha$ are two elements of $\mathcal{A}$, we have

$$xy = (x_0 + x_1\alpha)(y_0 + y_1\alpha) = x_0y_0 + (x_0y_1 + x_1y_0)\alpha.$$

We can again construct the matrix B and A, and solve $AX = B$, this time simply over $\mathbb{Q}$. We obtain

$$B = -\mathbf{f}(1) + 2^{-1}\mathbf{f}(1+\alpha) + 2^{-1}\mathbf{f}(1-\alpha)$$

so that the trisymmetric bilinear complexity of $\mathcal{A} = \mathbb{F}_q[T]/(T^2)$ is at most 3, which allows us to conclude. □

Proposition 4.2.4. *For any q not divisible by 2 nor 3 we have*

$$\hat{\mu}_q^{tri}(3) = 5 \qquad \textit{and} \qquad \hat{\mu}_q^{tri}(4) = 7.$$

Proof. We use the same notations as before. For $\mathcal{A} = \mathbb{F}_q[T]/(T^3)$, we obtain

$$B = -\mathbf{f}(1-\alpha-\alpha^2) + 3^{-1}\mathbf{f}(\alpha+2\alpha^2) + 2^{-1}\mathbf{f}(1-\alpha-2\alpha^2) - 3^{-1}\mathbf{f}(\alpha-\alpha^2) + 2^{-1}\mathbf{f}(1-\alpha).$$

Therefore the trisymmetric bilinear complexity of $\mathcal{A} = \mathbb{F}_q[T]/(T^3)$ is 5.

Finally, for $\mathcal{A} = \mathbb{F}_q[T]/(T^4)$, we obtain

$$\begin{aligned} B = \; & 2^{-1}\mathbf{f}(1-\alpha^2+\alpha^3) - \mathbf{f}(1-\alpha^2) + 12^{-1}\mathbf{f}(\alpha+2\alpha^2+2\alpha^3) - 12^{-1}\mathbf{f}(\alpha-2\alpha^2+2\alpha^3) \\ & - 6^{-1}\mathbf{f}(\alpha+\alpha^2-\alpha^3) + 6^{-1}\mathbf{f}(\alpha-\alpha^2-\alpha^3) + 2^{-1}\mathbf{f}(1-\alpha^2-\alpha^3)\cdot(1-\alpha^2-\alpha^3). \end{aligned}$$

The trisymmetric bilinear complexity of $\mathcal{A} = \mathbb{F}_q[T]/(T^4)$ is then 7. □

4.3 Asymptotic complexities

Parallel to the question of finding the hypersymmetric complexity of bilinear maps over $\mathbf{k}$-vector spaces of small dimension, it is also of theoretical importance to understand how these quantities asymptotically grow. In the bilinear case, we have rather precise bounds on

$$\limsup_{k\to\infty}\left(\frac{1}{k}\mu_q(k)\right).$$

In particular, these bounds imply that the bilinear complexity $\mu_q(k)$ grows linearly in k. Since we have similar results for the symmetric bilinear complexity, we also know that $\mu_q^{\mathrm{sym}}(k)$ grows linearly in k. A natural question that follows is then if we can also prove that the trisymmetric complexity

$$\mu_q^{\mathrm{tri}}(k)$$

still grows linearly in k. In this section, we work with either the algebra

$$\mathcal{A} = \mathbb{F}_{q^k} \text{ or } \mathcal{A} = \mathbb{F}_q[T]/(T^k),$$

seen as algebras over $\mathbf{k} = \mathbb{F}_q$, and equipped with the scalar product

$$\langle x, y\rangle = \tau(xy)$$

introduced in Section 4.1.4, *i.e.*

- if $\mathcal{A} = \mathbb{F}_{q^k}$, $\tau = \mathrm{Tr}$ is the trace map;
- if $\mathcal{A} = \mathbb{F}_q[T]/(T^k)$ and $x = \sum_{j=0}^{k-1} x_j T^j$, then $\tau(x) = x_{k-1}$ is the map sending an element to its leading coefficient.

Our aim is to show that the trisymmetric bilinear complexities $\mu_q^{\mathrm{tri}}(k)$ and $\hat{\mu}_q^{\mathrm{tri}}(k)$ grow linearly as $k \to \infty$. Our proof will involve higher multilinear maps, and in turn, give results for them as well.

For any s we define the s-multilinear multiplication map in $\mathcal{A}$ over $\mathbf{k}$

$$\begin{array}{rccc} m_s : & \mathcal{A}^s & \to & \mathcal{A} \\ & (x_1,\ldots,x_s) & \mapsto & x_1\cdots x_s \end{array}$$

and the s-multilinear trace form

$$\begin{array}{rccc} \tau_s = \tau \circ m_s : & \mathcal{A}^s & \to & \mathcal{A} \\ & (x_1, \dots, x_s) & \mapsto & \tau(x_1 \cdots x_s). \end{array}$$

If needed, we will write ${m_s}^{\mathcal{A}/\mathbf{k}}$ or ${\tau_s}^{\mathcal{A}/\mathbf{k}}$ to keep $\mathcal{A}$ and $\mathbf{k}$ explicit. The (symmetric) multilinear complexity of m_s has been considered in [Bsh13].

Lemma 4.3.1. *The map m_s is hypersymmetric, and we have*

$$\mu^{hyp}(m_s) = \mu^{sym}(\tau_{s+1}) \leq \mu^{sym}(m_{s+1}).$$

Proof. Recall that we have

$$\begin{aligned} \widetilde{m}_s(x_1, \dots, x_{s+1}) &\stackrel{\text{def}}{=} \langle m_s(x_1, \dots, x_s), x_{s+1} \rangle \\ &= \langle x_1 \dots x_s, x_{s+1} \rangle \\ &= \tau(x_1 \dots x_{s+1}) \\ &= \tau_{s+1}(x_1, \dots, x_{s+1}), \end{aligned}$$

and thus the equality on the left is a special case of Lemma 4.1.9. For the inequality on the right, assume we have a symmetric formula of length n for m_{s+1}, such that for all $x_1, \dots, x_{s+1} \in \mathcal{A}$, we have

$$m_{s+1}(x_1, \dots, x_{s+1}) = x_1 \dots x_{s+1} = \sum_{j=1}^{n} \langle a_j, x_1 \rangle \dots \langle a_j, x_{s+1} \rangle \, b_j$$

and apply τ. We obtain

$$\tau(x_1 \dots x_{s+1}) = \sum_{j=1}^{n} \langle a_j, x_1 \rangle \dots \langle a_j, x_{s+1} \rangle \, \tau(b_j),$$

that is precisely a symmetric decomposition for τ_{s+1} so the right inequality is proven. □

When studying the variation with the degree of the extension field $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$, we will write $\mu_q^{\text{sym}}(k, m_s)$ for $\mu^{\text{sym}}\left({m_s}^{\mathbb{F}_{q^k}/\mathbb{F}_q}\right)$, and we will also use the similar notations $\mu_q^{\text{hyp}}(k, m_s)$, $\mu_q^{\text{sym}}(k, \tau_s)$, etc. In particular for $s = 2$ we have

$$\mu_q^{\text{tri}}(k) \stackrel{\text{def}}{=} \mu_q^{\text{hyp}}(k, m_2) = \mu_q^{\text{sym}}(k, \tau_3).$$

When working in $\mathbb{F}_q[T]/(T^k)$ over $\mathbb{F}_q$, we will write likewise $\hat{\mu}_q^{\text{sym}}(k, m_s)$, $\hat{\mu}_q^{\text{hyp}}(k, m_s)$, etc.

Our aim is, for fixed q and s with $q \geq s + 1$, to show that $\mu_q^{\text{hyp}}(k, m_s)$ and $\hat{\mu}_q^{\text{hyp}}(k, m_s)$ grow linearly with $k \to \infty$. This is thus a stronger result than the proof that only the trisymmetric complexity, *i.e.* $s = 2$, grows linearly in the degree of the extension. Thanks to Lemma 4.3.1, it suffices to show that $\mu_q^{\text{sym}}(k, m_{s+1})$ and $\hat{\mu}_q^{\text{sym}}(k, m_{s+1})$ grow linearly with $k \to \infty$. Again, this is a generalization of the case $s = 2$ for which we already know that the symmetric bilinear complexity is linear in the dimension of the considered vector space. To ease notations we will set

$$M_{q,s}^{\text{sym}} = \limsup_{k \to \infty} \frac{1}{k} \mu_q^{\text{sym}}(k, m_s),$$

$$M_{q,s}^{\text{hyp}} = \limsup_{k \to \infty} \frac{1}{k} \mu_q^{\text{hyp}}(k, m_s),$$

$$M_q^{\text{tri}} = \limsup_{k \to \infty} \frac{1}{k} \mu_q^{\text{tri}}(k) = M_{q,2}^{\text{hyp}}$$

and likewise for $\hat{M}_{q,s}^{\text{sym}}$, $\hat{M}_{q,s}^{\text{hyp}}$, $\hat{M}_q^{\text{tri}}$, etc.

Evaluation-interpolation method. We use the function field terminology and notations presented in [Sti09] and recalled in Section 2.2. Let $F/\mathbb{F}_q$ be an algebraic function field of one variable over $\mathbb{F}_q$ and let $\mathbb{P}_F$ be the set of places of F. Let $\mathcal{D}_F$ be the set of divisors on F, and if $D \in \mathcal{D}_F$ is a divisor on F, we denote by $L(D)$ its Riemann-Roch space and we let $\ell(D) = \dim L(D)$ be the dimension of this space. Proposition 4.3.2 generalizes the idea of the Chudnovsky brothers to the product of s variables and highlights the method to obtain symmetric formulas.

Proposition 4.3.2. *Assume there exist a place $Q \in \mathbb{P}_F$ of F of degree k, $P_1, \ldots, P_n \in \mathbb{P}_F$ places of F of degree 1, and a divisor $D \in \mathcal{D}_F$ of F such that the places Q and $P_1, \ldots, P_n$ are not in the support of D and such that the following conditions hold.*

(i) The evaluation map

$$\begin{array}{rccc} \mathrm{ev}_{Q,D}: & L(D) & \to & \mathbb{F}_{q^k} \\ & f & \mapsto & f(Q) \end{array}$$

is surjective.

(ii) The evaluation map

$$\begin{array}{rccc} \mathrm{ev}_{\mathcal{P},sD}: & L(sD) & \to & (\mathbb{F}_q)^n \\ & h & \mapsto & (h(P_1), \ldots, h(P_n)) \end{array}$$

is injective.

Then $m_s^{\mathbb{F}_{q^k}/\mathbb{F}_q}$ admits a symmetric formula of length n, i.e. *we have $\mu_q^{sym}(k, m_s) \leq n$.*

Proof. Since the map $\mathrm{ev}_{Q,D}$ is surjective, it admits a right inverse, *i.e.* a linear map

$$u : \mathbb{F}_{q^k} \to L(D)$$

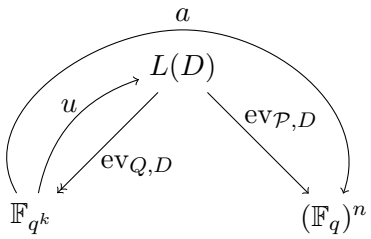
such that

$$\mathrm{ev}_{Q,D} \circ u = \mathrm{Id}_{\mathbb{F}_{q^k}}.$$

For all $x \in \mathbb{F}_{q^k}$, we denote $u(x) \in L(D)$ by f_x, so the map $x \mapsto f_x$ is linear, and $f_x(Q) = x$. We also let

$$\begin{array}{rccc} a: & \mathbb{F}_{q^k} & \to & (\mathbb{F}_q)^n \\ & x & \mapsto & (f_x(P_1), \ldots, f_x(P_n)) \end{array}$$

be the composite map $a = \mathrm{ev}_{\mathcal{P},D} \circ u$. The situation is summed up in the following diagram.



Observe that a is linear, so we can write

$$a(x) = (\varphi_1(x), \ldots, \varphi_n(x))$$

where $\varphi_i : \mathbb{F}_{q^k} \to \mathbb{F}_q$ is a linear form, namely $\varphi_i(x) = f_x(P_i)$.

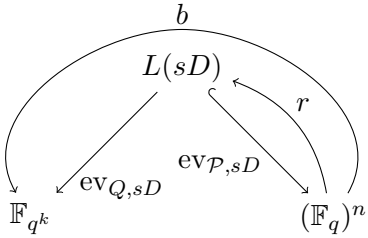
Similarly, since the map $\mathrm{ev}_{\mathcal{P},sD}$ is injective, it admits a left inverse, *i.e.* a linear map

$$r : (\mathbb{F}_q)^n \to L(sD)$$

such that

$$r \circ \mathrm{ev}_{\mathcal{P},sD} = \mathrm{Id}_{L(sD)}\,.$$

We also let $b : (\mathbb{F}_q)^n \to \mathbb{F}_{q^k}$ be the composite map $b = \mathrm{ev}_{Q,sD} \circ r$. The situation is summed up in the following diagram.



The map b is linear, so there are $b_1, \ldots, b_n$ in $\mathbb{F}_{q^k}$ such that, for all $y = (y_1, \ldots, y_n) \in (\mathbb{F}_q)^n$,

$$b(y) = \sum_{i=1}^{n} y_i b_i.$$

Now for $x, \ldots, x_s \in \mathbb{F}_{q^k}$, let

$$p = (p_1, \ldots, p_n) = ((\prod_{j=1}^{s} f_{x_j})(P_1), \ldots, (\prod_{j=1}^{s} f_{x_j})(P_n))$$

in $(\mathbb{F}_q)^n$ be the coordinatewise product of the vectors $a(x_1)$, ..., $a(x_s)$. Then

$$h = r(p)$$

is an element of $L(sD)$ such that $h(P_i) = p_i = (\prod_{j=1}^{s} f_{x_j})(P_i)$ for all i. Since the map $\mathrm{ev}_{\mathcal{P},sD}$ is injective, this forces

$$h = \prod_{j=1}^{s} f_{x_j}.$$

Then, we have

$$b(p) = \mathrm{ev}_{Q,sD}(r(p)) = \mathrm{ev}_{Q,sD}(h) = h(Q) = \prod_{j=1}^{s} f_{x_j}(Q) = \prod_{j=1}^{s} x_j.$$

But we also have

$$b(p) = \sum_{i=1}^{n} p_i b_i = \sum_{i=1}^{n} (\prod_{j=1}^{s} f_{x_j}(P_i)) b_i = \sum_{i=1}^{n} (\prod_{j=1}^{s} \varphi_i(x_j)) b_i$$

and finally we get a symmetric formula for m_s:

$$\prod_{j=1}^{s} x_j = \sum_{i=1}^{n} (\prod_{j=1}^{s} \varphi_i(x_j)) b_i.$$

□

Now that we have a method to find formulas, we have to prove that Conditions (i) and (ii) can be satisfied. Proposition 4.3.3 gives sufficient assumptions to obtain these conditions.

Proposition 4.3.3. *Let $F/\mathbb{F}_q$ be an algebraic function field of genus g. Assume that F admits a place Q of degree k, and a set $\mathcal{S}$ of places of degree 1 of size*

$$|\mathcal{S}| \geq (k+g-1)s+1.$$

Then we have

$$\mu_q^{sym}(k, m_s) \leq ks + (g-1)(s-1).$$

Proof. Set $n = ks + (g-1)(s-1)$. We will show that there are places $P_1, \ldots, P_n$ in $\mathcal{S}$, and a divisor D on F, such that Proposition 4.3.2 applies, which gives $\mu_q^{\mathrm{sym}}(k, m_s) \leq n$ as desired.

Using e.g. [Bal99, Lemma 2.1] we know F admits a non-special divisor R of degree $g-1$. By the strong approximation theorem [Sti09, Thm. 1.6.5] we can then find a divisor D linearly equivalent to $R+Q$ and of support disjoint from Q and $\mathcal{S}$.

Then $D-Q$ and D are non-special, with $\ell(D-Q)=0$ and $\ell(D)=k$. We thus find

$$\mathrm{Ker}(\mathrm{ev}_{Q,D} : L(D) \to \mathbb{F}_{q^k}) = L(D-Q) = 0,$$

so $\mathrm{ev}_{Q,D}$ is injective, hence also surjective by equality of dimensions, *i.e.* the surjectivity condition (i) in Proposition 4.3.2 is satisfied.

Likewise, sD is non-special, with $\deg(sD) = (k+g-1)s$ and $\ell(sD) = ks + (g-1)(s-1)$. Then the evaluation map

$$\begin{array}{rccc} \mathrm{ev}_{\mathcal{S},sD} : & L(sD) & \to & (\mathbb{F}_q)^{|\mathcal{S}|} \\ & h & \mapsto & (h(P))_{P\in\mathcal{S}} \end{array}$$

has kernel $L(sD - \sum_{P\in\mathcal{S}} P) = 0$, because $\deg(sD - \sum_{P\in\mathcal{S}} P) = (k+g-1)s - |\mathcal{S}| < 0$. So $\mathrm{ev}_{\mathcal{S},sD}$ is injective, with image of dimension $\dim \mathrm{Im}(\mathrm{ev}_{\mathcal{S},sD}) = \ell(sD) = n$. Then we can find a subset $\mathcal{P} = \{P_1, \ldots, P_n\} \subset \mathcal{S}$ of size n, such that $\mathrm{ev}_{\mathcal{P},sD} : L(sD) \to (\mathbb{F}_q)^n$ is an isomorphism, and the injectivity condition (ii) in Proposition 4.3.2 is also satisfied. □

Choice of the curves for q a large enough square. Now that we have somewhat easier assumptions to fulfill, that are only based on the existence of a certain number of places, we prove that we can indeed find algebraic function fields that satisfy these properties. We first prove it in the special case, where the size q of the base field $\mathbf{k}$ is a large enough square.

Proposition 4.3.4. *Let s be given, and assume q is a square, $q \geq (s+2)^2$. Then we have*

$$M_{q,s}^{sym} \leq (1+\epsilon_s(q))s$$

with $\epsilon_s(q) = \frac{s-1}{\sqrt{q}-s-1}$.

Proof. We know [STV92] that there exists a family of function fields $F_i/\mathbb{F}_q$ of genus $g_i \to \infty$ such that

(i) $\frac{g_{i+1}}{g_i} \to 1$

(ii) $N_i \sim (\sqrt{q}-1)g_i$

where $N_i = \#\{P \in \mathbb{P}_{F_i} \mid \deg P = 1\}$ is the number of places of degree 1 of F_i. We can also assume that the sequence g_i is increasing.

For any k let $i(k)$ be the smallest index such that

$$N_{i(k)} \geq (k + g_{i(k)} - 1)s + 1.$$

Such an $i(k)$ always exists since by (ii) we have $N_i \sim (\sqrt{q} - 1)g_i$, with $\sqrt{q} - 1 > s$.

By definition we thus have

$$N_{i(k)} \geq (k + g_{i(k)} - 1)s + 1 > (k + g_{i(k)-1} - 1)s + 1 > N_{i(k)-1}.$$

As $k \to \infty$ we have $i(k) \to \infty$, and by (i) we get

$$g_{i(k)} \sim g_{i(k)-1},$$

so by (ii) we also get

$$N_{i(k)} \sim N_{i(k)-1}.$$

This then gives

$$\begin{aligned} N_{i(k)} &\sim (k + g_{i(k)} - 1)s + 1 \\ &\sim (k + g_{i(k)})s \end{aligned}$$

while by (ii),

$$N_{i(k)} \sim (\sqrt{q} - 1)g_{i(k)}.$$

From these two relations we deduce

$$g_{i(k)} \sim \frac{s}{\sqrt{q} - 1 - t}k.$$

For k large enough this implies in particular $2g_{i(k)} + 1 \leq q^{(k-1)/2}(\sqrt{q} - 1)$, so $F_{i(k)}$ admits a place of degree k by [Sti09, Cor. 5.2.10].

From this we are allowed to apply Proposition 4.3.3 to $F_{i(k)}$, which gives

$$\mu_q^{\mathrm{sym}}(k, m_s) \ \leq \ ks + (g_{i(k)} - 1)(s - 1) \ \sim \ ks + g_{i(k)}(s - 1) \ \sim \ ks(1 + \epsilon_s(q))$$

as desired. □

Corollary 4.3.5. *For q a square, $q \geq (s + 3)^2$ we have*

$$M_{q,s}^{hyp} \leq (1 + \epsilon_{s+1}(q))(s + 1),$$

with $\epsilon_s(q) = \frac{s-1}{\sqrt{q}-s-1}$, and in particular we have

$$M_q^{tri} \leq 3\left(1 + \frac{2}{\sqrt{q} - 4}\right)$$

for q a square, $q \geq 25$.

Conclusion for arbitrary q. Finally, we complete our objective of proving that the symmetric multilinear complexity is linear in the degree of the extension by extending the previous result to arbitrary q.

Lemma 4.3.6. *Let q be a prime power. Then for any integers s, d, k we have*

$$\mu_q^{sym}(k, m_s) \leq \mu_q^{sym}(dk, m_s) \leq \mu_q^{sym}(d, m_s)\mu_{q^d}^{sym}(k, m_s).$$

Proof. For the inequality on the left, there is nothing to prove if $\mu_q^{\mathrm{sym}}(dk, m_s) = \infty$. So let us assume $m_s^{\mathbb{F}_{q^{dk}}/\mathbb{F}_q}$ admits a symmetric multiplication formula of length $n = \mu_q^{\mathrm{sym}}(dk, m_s)$, *i.e.*

$$\forall x_1, \ldots, x_s \in \mathbb{F}_{q^{dk}}, \quad x_1 \cdots x_s = \sum_{i=1}^{n} \varphi_i(x_1) \cdots \varphi_i(x_s) a_i$$

for linear forms $\varphi_i : \mathbb{F}_{q^{dk}} \to \mathbb{F}_q$ and elements $a_i \in \mathbb{F}_{q^{dk}}$. Choose a linear projection

$$p : \mathbb{F}_{q^{dk}} \to \mathbb{F}_{q^k}$$

left inverse for the inclusion $\mathbb{F}_{q^k} \subseteq \mathbb{F}_{q^{dk}}$. Then we get

$$\forall x_1, \ldots, x_s \in \mathbb{F}_{q^k}, \quad x_1 \cdots x_s = p(x_1, \ldots, x_s) = \sum_{i=1}^{n} \varphi_i(x_1) \cdots \varphi_i(x_s) p(a_i)$$

which is a symmetric multiplication formula of length n for $m_s^{\mathbb{F}_{q^k}/\mathbb{F}_q}$.

Likewise, for the inequality on the right, there is nothing to prove if $\mu_q^{\mathrm{sym}}(d, m_s) = \infty$ or $\mu_{q^d}^{\mathrm{sym}}(k, m_s) = \infty$. So let us assume $m_s^{\mathbb{F}_{q^d}/\mathbb{F}_q}$ and $m_s^{\mathbb{F}_{q^{dk}}/\mathbb{F}_{q^d}}$ admit symmetric multiplication formulas of length $\rho = \mu_q^{\mathrm{sym}}(d, m_s)$ and $\mu = \mu_{q^d}^{\mathrm{sym}}(k, m_s)$ respectively, so

$$\forall y_1, \ldots, y_s \in \mathbb{F}_{q^d}, \quad y_1 \cdots y_s = \sum_{u=1}^{\rho} \psi_u(y_1) \cdots \psi_u(y_s) b_u$$

$$\forall z_1, \ldots, z_s \in \mathbb{F}_{q^{dk}}, \quad z_1 \cdots z_s = \sum_{v=1}^{\mu} \chi_v(z_1) \cdots \chi_v(z_s) c_v$$

for linear forms $\psi_u : \mathbb{F}_{q^d} \to \mathbb{F}_q$, $\chi_v : \mathbb{F}_{q^{dk}} \to \mathbb{F}_{q^d}$ and elements $b_u \in \mathbb{F}_{q^d}$, $c_v \in \mathbb{F}_{q^{dk}}$. Then setting $y_1 = \chi_v(z_1)$, ..., $y_s = \chi_v(z_s)$ we find

$$\forall z_1, \ldots, z_s \in \mathbb{F}_{q^{dk}}, \quad z_1 \cdots z_s = \sum_{v=1}^{\mu} \sum_{u=1}^{\rho} (\psi_u \circ \chi_v)(z_1) \cdots (\psi_u \circ \chi_v)(z_s) \cdot (b_u c_v)$$

which is a symmetric multiplication formula of length $\rho\mu$ for $m_s^{\mathbb{F}_{q^{dk}}/\mathbb{F}_q}$. □

Theorem 4.3.7. *Let $s \geq 2$ be an integer and q a prime power. If $q < t$, then $\mu_q^{sym}(k, m_s) = \infty$ for all $k \geq 2$.*

On the other hand, if $q \geq s$, then $\mu_q^{sym}(k, m_s)$ grows at most linearly with k, i.e. *we have*

$$M_{q,s}^{sym} \leq C_s(q)$$

for some real constant $C_s(q) < \infty$.

Proof. If $q < t$ and $k \geq 2$, then $\mu_q^{\mathrm{sym}}(k, m_s) = \infty$ follows from Theorem 4.1.5.

On the other hand, for $q \geq s$, we have $\mu_q^{\mathrm{sym}}(d, m_s) < \infty$ for any integer d. Choose d such that q^d is a square, $q^d \geq (s+2)^2$. Then Proposition 4.3.4 shows $\mu_{q^d}^{\mathrm{sym}}(k, m_s)$ grows linearly with k. The Theorem then follows thanks to Lemma 4.3.6, with $C_s(q) = \mu_q^{\mathrm{sym}}(d, m_s)(1 + \epsilon_s(q^d))s$. □

Corollary 4.3.8. *For $q \geq s+1$ we have*

$$M_{q,s}^{hyp} \leq C_{s+1}(q)$$

and in particular for $q \geq 3$ we have

$$M_q^{tri} \leq C_3(q).$$

Part II

Efficient arithmetic in a lattice of finite fields

Chapter 5

Isomorphism algorithms

We studied in Part I the arithmetic of a single finite field extension. We now study a set of several extensions. The very first step will be to understand how to compute an isomorphism (or an embedding) between two finite fields: this is the material of this chapter.

Contents

5.1 Preliminaries and naive algorithm 90
5.1.1 Description of the problem 90
5.1.2 Embedding description problem and naive algorithm 92
5.2 Lenstra-Allombert algorithm 92
5.2.1 Preliminaries 92
5.2.2 Kummer algebras 96
5.2.3 The isomorphism algorithm 101
5.2.4 Computing (H90) solutions 102
5.3 The embedding evaluation problem 103
5.3.1 Linear algebra 104
5.3.2 Inverse maps and duality 104
5.3.3 Modular composition 108

Our reference for this chapter is [BDFD+17]: although it provides a variety of isomorphism algorithms (and their analysis), we are mainly interested in the naive isomorphism algorithm (used in Chapter 6) and in Allombert's algorithm (used in Chapter 7). We thus do not cover all the isomorphism algorithms, the reader interested in Rains' algorithm [Pin92, Rai96] and its elliptic variant can take a look at the paper cited above.

5.1 Preliminaries and naive algorithm

Even if our real goal is to compute *embeddings* of finite fields, *i.e.* ring homomorphisms

$$\phi : K \to L$$

with K and L finite fields, we often refer to the algorithms as *isomorphism* algorithms. Indeed, computing the embedding ϕ is the same as computing an isomorphism

$$\phi' : K \xrightarrow{\sim} K'$$

of K with a subfield $K' \subset L$ of L. The isomorphism ϕ' is just the embedding ϕ with its codomain being restricted to K'. That is why in the remainder of this chapter, we present *isomorphism* algorithms, rather than embedding algorithms.

5.1.1 Description of the problem

We let p be a prime number, $\mathbf{k} = \mathbb{F}_p$ be the field with p elements, and $f, g \in \mathbf{k}[X]$ two irreducible polynomials of degree $m = \deg(f)$ and $n = \deg(g)$, with

$$m \mid n.$$

Let

$$K = \mathbf{k}[X]/(f(X)) \cong \mathbb{F}_{p^m}$$

and

$$L = \mathbf{k}[Y]/(g(Y)) \cong \mathbb{F}_{p^n}$$

be two extensions of $\mathbf{k}$. We know there is an embedding

$$\phi : K \to L,$$

unique up to $\mathbf{k}$-automorphism of K, *i.e.* there are

$$\#\operatorname{Gal}(K/\mathbf{k}) = m$$

different embeddings from K to L, that can be described as

$$\phi \circ \sigma$$

for $\sigma \in \operatorname{Gal}(K/\mathbf{k})$. Equivalently, they can also be described as

$$\sigma' \circ \phi$$

with $\sigma' \in \operatorname{Gal}(\phi(K)/\mathbf{k})$. The *embedding problem* is then to efficiently find, represent and evaluate one such embedding ϕ. Following [BDFD+17], the problem is split in two parts.

Embedding description problem. Compute elements $\alpha \in K$ and $\beta \in L$ such that

$$K = \mathbf{k}(\alpha)$$

and such that there exists an embedding ϕ mapping α to β.

Embedding evaluation problem. Given elements α and β defined above, and elements $x \in K$, $y \in L$, solve the following problems:

- compute $\phi(x) \in L$;
- test if $y \in \phi(K)$;
- if $y \in \phi(K)$, then compute $\phi^{-1}(y) \in K$.

As the name suggests, the *embedding description problem* focuses on finding a pair of elements that are sufficient to describe an embedding. Indeed, if

$$K = \mathbf{k}(\alpha)$$

we know that every element $x \in K$ can be uniquely written as

$$x = \sum_{j=0}^{m-1} a_j \alpha^j$$

with $a_j \in \mathbf{k}$ for al $0 \leq j \leq m-1$, and the embedding ϕ is then defined by

$$\phi(x) = \sum_{j=0}^{m-1} a_j \beta^j.$$

Proposition 5.1.1. *The elements α and β describe an embedding if and only if they have the same minimal polynomial over $\mathbf{k}$.*

Proof. Let $\phi : K \to L$ be an embedding mapping α to β and let

$$P = \mathrm{Minpoly}_{\mathbf{k}}(\alpha)$$

be the the minimal polynomial of α. Then

$$\begin{aligned} P(\beta) &= P(\phi(\alpha)) \\ &= \phi(P(\alpha)) \\ &= \phi(0) \\ &= 0 \end{aligned}$$

thus $\mathrm{Minpoly}_{\mathbf{k}}(\beta) \neq 1$ divides P which is irreducible so

$$\mathrm{Minpoly}_{\mathbf{k}}(\beta) = P.$$

Conversely, if α and β have the same minimal polynomial P, then the map ϕ is well-defined and defines an isomorphism between the fields $\mathbf{k}(\alpha)$ and $\mathbf{k}(\beta)$, that are both isomorphic to the field

$$\mathbf{k}[X]/(P(X)).$$

□

While the first problem focuses on finding a description of ϕ, the *embedding evaluation problem* independently asks how to efficiently use the description to compute the actual embedding. We target this question in Section 5.3.

5.1.2 Embedding description problem and naive algorithm

Until the end of this section and in Section 5.2, we deal with the *embedding description problem*, although we only review a subpart of the existing algorithms (see [BDFD+17] for other algorithms). As above, let f and g be two irreducible polynomials with coefficients in $\mathbf{k}$ and with respective degree m and n, such that

$$m \mid n.$$

Let

$$K = \mathbf{k}[X]/(f(X)) \cong \mathbb{F}_{p^m}$$

and

$$L = \mathbf{k}[Y]/(g(Y)) \cong \mathbb{F}_{p^n}$$

be two finite fields. Then one can simply take α to be the class of X in K and choose β to be any root of f in L. Indeed, we know that there is an isomorphic copy of K in L and thus that f splits over L. Furthermore, any root of f will have f as its minimal polynomial, which is also the minimal polynomial of α by construction. By Proposition 5.1.1, the map

$$\phi : K \to L$$

sending α to β is an embedding. The critical routine in that algorithm is to find a root of f in L, that can be done using the Shoup-Kaltofen *equal degree factorization* algorithm [KS97]. The complexity analysis of [BDFD+17] indicates that the cost is strictly larger than quasi-quadratic complexity $\tilde{O}(m^2)$. A more efficient algorithm, due to Lenstra and Allombert, is discussed in Section 5.2.

5.2 Lenstra-Allombert algorithm

Both Lenstra [LJ91] and Allombert used Kummer theory, the study of certain field extensions, to compute isomorphisms between finite fields. But while Lenstra's focus was on proving the existence of a deterministic isomorphism algorithm, Allombert wanted to provide a practical algorithm. This led to the invention of the Lenstra-Allombert algorithm [All02a] in 2002, for which we give a description in this section. The ideas of Allombert play an important part in Chapter 7 too. The techniques based on Kummer theory work for extensions of degree n coprime to the characteristic p. In order to have an algorithm working for any type of extension, the solution is to deal with the part of the extension which degree is divisible by p separately using Artin-Shreier theory and to glue the results together in the end. More details can be found in [BDFD+17, Section 3.2].

5.2.1 Preliminaries

Let us first discuss a simpler case than the general one, that will highlight the method behind the Lenstra-Allombert isomorphism algorithm. Let K and L be two finite fields of cardinality p^n, such that

$$K \cong L \cong \mathbb{F}_{p^n}.$$

Assume that $\gcd(p, n) = 1$ and that

$$n \mid p - 1,$$

or equivalently that there is a primitive n-th root of unity in $\mathbf{k} = \mathbb{F}_p$, that we denote by ζ. The algorithm is based on Proposition 5.2.1.

Proposition 5.2.1 (Hilbert 90 theorem)**.** *Let K be a finite extension of $\mathbf{k} = \mathbb{F}_p$ of degree n such that there exists a primitive n-th root of unity $\zeta \in \mathbf{k}$ in the base field $\mathbf{k}$,* i.e. *such that n divides $p-1$. Let σ be the Frobenius automorphism of the extension*

$$K/\mathbf{k}$$

and consider the following equation in K:

$$\sigma(x) = \zeta x. \tag{H90}$$

The solutions of (H90) *form a one dimensional $\mathbf{k}$-vector space and if $\alpha \in K$ is such a solution, we have*

$$\alpha^n \in \mathbf{k}.$$

If α is also nonzero, then it is a generator of K over $\mathbf{k}$.

Proof. Let us first construct a nonzero solution of (H90). Consider the polynomial

$$P = \sum_{j=0}^{n-1} \zeta^{-j} X^{p^j}$$

of degree p^{n-1}. The polynomial P has at most p^{n-1} roots in K, which has cardinality p^n, so there exists some element $x \in K$ such that

$$y = P(x) \neq 0.$$

Now, by construction, we have

$$\begin{aligned}
\sigma(y) &= \sigma(\sum_{j=0}^{n-1} \zeta^{-j} x^{\sigma^j}) \\
&= \sum_{j=0}^{n-1} \zeta^{-j} x^{\sigma^{j+1}} \\
&= \zeta \times \sum_{j=0}^{n-1} \zeta^{-(j+1)} x^{\sigma^{j+1}} \\
&= \zeta \times \sum_{j=1}^{n} \zeta^{-j} x^{\sigma^j} \\
&= \zeta y
\end{aligned}$$

and thus y is a nonzero solution of (H90). All the elements

$$\lambda y$$

with $\lambda \in \mathbf{k}$ are also solutions of (H90) since

$$\sigma(\lambda y) = \lambda \sigma(y) = \zeta \lambda y,$$

and the equation has at most p solutions because the polynomial

$$X^p - \zeta X$$

has at most p roots in K. Thus there are exactly p different solutions, that are the elements of $\mathrm{Vect}(y)$. Let z be a solution of (H90), then we have

$$\begin{aligned}\sigma(z^n) &= \sigma(z)^n \\ &= (\zeta z)^n \\ &= z^n,\end{aligned}$$

therefore z^n is fixed by σ, which means that

$$z^n \in \mathbf{k}.$$

If z is also nonzero, then for all $0 \leq j < n$, we have

$$\begin{aligned}\sigma^j(z) &= \underbrace{(\sigma \circ \cdots \circ \sigma)}_{j \text{ times}}(z) \\ &= \underbrace{(\sigma \circ \cdots \circ \sigma)}_{j-1 \text{ times}}(\zeta z) \\ &= \zeta \underbrace{(\sigma \circ \cdots \circ \sigma)}_{j-1 \text{ times}}(z) \\ &= \zeta^j z \\ &\neq z.\end{aligned}$$

Consequently, z is not in any subfield of K and is thus a generator of K over $\mathbf{k}$. □

Note that Proposition 5.2.1 applies both to the fields K and L, therefore we can solve Equation (H90) in both fields. Let α_K be a solution of Equation (H90) for the root ζ in K, and α_L a solution in L. Since we want the primitive n-th root of unity ζ to be the same in K and L, we assume that we already have an embedding from $\mathbf{k}$ in both these fields. In practice, since $\mathbf{k}$ is a prime field and the fields K and L are represented by polynomials over $\mathbf{k} = \mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$, the assumption is not really hard to meet. Let

$$a_K = \alpha_K^n$$

and

$$a_L = \alpha_L^n.$$

By Proposition 5.2.1, we know that a_K and a_L are both in $\mathbf{k}$ and this can be used to compute an isomorphism between K and L.

Proposition 5.2.2 (Allombert [All02a])**.** *The quotient*

$$a_K/a_L$$

is an n-th power in $\mathbf{k}$, and if

$$c^n = a_K/a_L$$

then the map sending α_K to $c\alpha_L$ is an isomorphism from K to L.

Proof. Let $\phi : K \to L$ be a $\mathbf{k}$-isomorphism between K and L. We have

$$\begin{aligned}\sigma(\phi(\alpha_K)) &= \phi(\alpha_K)^p \\ &= \phi(\alpha_K^p) \\ &= \phi(\sigma(\alpha_K)) \\ &= \phi(\zeta\alpha_K) \\ &= \zeta\phi(\alpha_K)\end{aligned}$$

thus $\phi(\alpha_K)$ is a solution of (H90) and by Proposition 5.2.1 there exists $\lambda \in \mathbf{k}$ such that

$$\phi(\alpha_K) = \lambda\alpha_L.$$

We also have

$$\phi(\alpha_K)^n = \phi(\alpha_K^n) = \phi(a_K) = a_K,$$

therefore the quotient

$$\begin{aligned}a_K/a_L &= \phi(\alpha_K)^n/\alpha_L^n \\ &= \lambda^n\end{aligned}$$

is an n-th power in $\mathbf{k}$. Now let $c \in \mathbf{k}$ be any n-th root of a_K/a_L, then

$$c = \zeta^j\lambda$$

for some $0 \leq j \leq n-1$, that is c and λ differ by a n-th root of unity, and so do $\phi(\alpha_K)$ and $c\alpha_L$:

$$c\alpha_L = \zeta^j\phi(\alpha_K) = \sigma^j(\phi(\alpha_K)).$$

Finally, the elements $c\alpha_L$ and $\phi(\alpha_K)$ have the same minimal polynomial, because they are conjugates, and $\phi(\alpha_K)$ has the same minimal polynomial as α_K because ϕ is an isomorphism, so by Proposition 5.1.1, the map sending α_K to $c\alpha_L$ is an isomorphism from K to L. □

In this simpler case (n divides $p-1$), Lenstra-Allombert algorithm consists in

1. finding $\alpha_K \in K$ and $\alpha_L \in L$ with $\sigma(\alpha_K) = \zeta\alpha_K$ and $\sigma(\alpha_L) = \zeta\alpha_L$;
2. computing a n-th root $c \in \mathbf{k}$ of α_K^n/α_L^n;
3. returning the isomorphism described by $\alpha_K \mapsto c\alpha_L$.

General case. When $n \nmid p-1$, which is always the case asymptotically since we work with p fixed and we let n grow, there are no n-th roots of unity in $\mathbf{k}$, and the strategy of Section 5.2.1 cannot be applied as if. Nevertheless, it is still possible to apply a similar idea by extending the space so that it contains artificial roots of unity.

5.2.2 Kummer algebras

Instead of "just" working in $\mathbb{F}_{p^n}$, we work in

$$A_n = \mathbb{F}_{p^n} \otimes \mathbb{F}_p(\zeta),$$

where ζ is a primitive n-th root of unity, and where $\otimes$ is the tensor product over $\mathbf{k} = \mathbb{F}_p$. We thus extend the scalars and force the existence of suitable roots of unity. The $\mathbf{k}$-algebra A_n can now be used instead of $\mathbb{F}_{p^n}$ in the Lenstra-Allombert algorithm. Following the terminology of [DFRR19], we call these algebras *Kummer algebras*.

Definition 5.2.3 (Kummer algebra)**.** Let $p \in \mathbb{N}$ a prime number and $\mathbf{k} = \mathbb{F}_p$ the finite field with p elements. We call the $\mathbf{k}$-algebra

$$A_n = \mathbb{F}_{p^n} \otimes \mathbb{F}_p(\zeta),$$

where $\otimes$ is the tensor product over $\mathbf{k}$, a *Kummer algebra of degree n*.

Definition 5.2.4 (Field of scalars)**.** Let A_n be a Kummer algebra of degree n. Then we define $\mathbb{F}_p(\zeta)$ as the *field of scalars* of A_n, and we define the *level* $\nu(n)$ of A_n as

$$\nu(n) = \mathrm{ord}_{(\mathbb{Z}/n\mathbb{Z})^\times}(p) = [\mathbb{F}_p(\zeta) : \mathbf{k}]\,,$$

where $\mathrm{ord}_{(\mathbb{Z}/n\mathbb{Z})^\times}(p)$ is the order of p in the multiplicative group $(\mathbb{Z}/n\mathbb{Z})^\times$. The level $\nu(n)$ of A_n is the degree of its field of scalars.

Let $\sigma : x \mapsto x^p$ be the Frobenius automorphism of the extension

$$\mathbb{F}_{p^n}/\mathbf{k},$$

and extend it to A_n by defining the linear map

$$\begin{array}{rccc} \sigma \otimes 1 : & A_n & \to & A_n \\ & \sum_j x_j \otimes y_j & \mapsto & \sum_j \sigma(x_j) \otimes y_j. \end{array}$$

The map $\sigma \otimes 1$ will play the role of σ in the simpler case.

Lemma 5.2.5. *The map $\sigma \otimes 1$ is a $1 \otimes \mathbb{F}_p(\zeta)$-linear endomorphism with n distinct eigenvalues, that are the powers of $1 \otimes \zeta$.*

Proof. Because of the linear independence of characters [Lan04, Chapter VI, §4], we know that the automorphisms

$$\mathrm{Id}, \sigma, \sigma^2, \ldots, \sigma^{n-1}$$

are independent. We also know that

$$\sigma^n = \mathrm{Id},$$

therefore the minimal polynomial of the $\mathbf{k}$-linear endomorphism σ is

$$X^n - 1.$$

By the Cayley-Hamilton theorem, we deduce that the characteristic polynomial of σ is also $X^n - 1$. Now let $\mathcal{B} = \{b_1, \ldots, b_n\}$ be a basis of $\mathbb{F}_{p^n}/\mathbf{k}$ and let M be the matrix of σ in this basis. Then the matrix of the $1 \otimes \mathbb{F}_p(\zeta)$-linear endomorphism $\sigma \otimes 1$ in the basis

$$\mathcal{B} \otimes 1 = \{b_1 \otimes 1, \ldots, b_n \otimes 1\}$$

is also M. Thus, the characteristic polynomial of $\sigma \otimes 1$ is again $X^n - 1$, that splits completely in

$$1 \otimes \mathbb{F}_p(\zeta) \cong \mathbb{F}_p(\zeta),$$

the roots being the elements

$$1 \otimes \zeta^j$$

for $0 \leq j \leq n-1$. Finally, the eigenvalues are the roots of the characteristic polynomial so this concludes the proof. □

Since there are exactly n distinct eigenvalues, we know that the corresponding eigenspaces are all one-dimensional $1 \otimes \mathbb{F}_p(\zeta)$-vector spaces, and the eigenspace corresponding to the eigenvalue ζ is described by the equation

$$(\sigma \otimes 1)(x) = (1 \otimes \zeta)x \tag{H90}$$

that we again denote by (H90). The field of scalars of A_n now plays the role of the base field $\mathbf{k}$ in the simpler case and the solutions of (H90) have similar properties.

Lemma 5.2.6. *The set of elements in A_n fixed by $\sigma \otimes 1$ is*

$$1 \otimes \mathbb{F}_p(\zeta) \cong \mathbb{F}_p(\zeta),$$

a subfield isomorphic to the field of scalars of A_n.

Proof. Let

$$\mathcal{B} = \{b_1, \ldots, b_a\}$$

be a basis of

$$\mathbb{F}_p(\zeta)/\mathbf{k}.$$

Then, every element α in A_n can be uniquely written in the form

$$\alpha = \sum_{j=1}^{a} x_j \otimes b_j$$

where for all $1 \leq j \leq a$, $x_j \in \mathbb{F}_{p^n}$. If

$$\begin{aligned} \sum_{j=1}^{a} \sigma(x_j) \otimes b_j &= (\sigma \otimes 1)(\alpha) \\ &= \alpha \end{aligned}$$

it follows that for all $1 \leq j \leq a$, we have

$$\sigma(x_j) = x_j$$

and thus we have $x_j \in \mathbf{k}$. Consequently, the element α belongs to the set

$$\mathbf{k} \otimes \mathbb{F}_p(\zeta) = 1 \otimes \mathbb{F}_p(\zeta) \cong \mathbb{F}_p(\zeta).$$

Conversely, if an element α is in $1 \otimes \mathbb{F}_p(\zeta)$, then it is fixed by $\sigma \otimes 1$. □

Remark 5.2.7. We can use the proof of Lemma 5.2.6, *mutatis mutandis*, in order to prove that the set of elements in A_n fixed by $1 \otimes \sigma'$, with σ' the Frobenius automorphism of the extension $\mathbb{F}_p(\zeta)/\mathbf{k}$, is $\mathbb{F}_{p^n} \otimes 1$. In the remainder of the document, we write σ for both σ and σ'.

Lemma 5.2.8. *Let α be a nonzero solution of (H90) for the root ζ. Then*

$$\alpha^n \in 1 \otimes \mathbb{F}_p(\zeta)$$

and α is a generating element for A_n as an algebra over $1 \otimes \mathbb{F}_p(\zeta)$ that is also invertible.

Proof. We have

$$\begin{aligned}(\sigma \otimes 1)(\alpha^n) &= (\sigma \otimes 1)(\alpha)^n \\ &= ((1 \otimes \zeta)\alpha)^n \\ &= (1 \otimes \zeta^n)\alpha^n \\ &= \alpha^n,\end{aligned}$$

thus $\alpha^n \in 1 \otimes \mathbb{F}_p(\zeta)$ by Lemma 5.2.6. Since α is a solution of (H90) for ζ, then for every $1 \leq j \leq n-1$, α^j is a solution for ζ^j, indeed

$$\begin{aligned}(\sigma \otimes 1)(\alpha^j) &= ((\sigma \otimes 1)(\alpha))^j \\ &= (1 \otimes \zeta^j)\alpha^j.\end{aligned}$$

Then, the elements $1, \alpha, \ldots, \alpha^{n-1}$ are eigenvectors for distinct eigenvalues of $\sigma \otimes 1$ and thus form a basis of A_n over $1 \otimes \mathbb{F}_p(\zeta)$. Assume that $\alpha^n = 1 \otimes c$ for some $c \in \mathbb{F}_p(\zeta)$. There are no nonzero nilpotent elements in A_n; one way of seeing it is to say that

$$A_n \cong \mathbb{F}_p(\zeta)[T]/(h(T))$$

where h is the irreducible polynomial defining

$$\mathbb{F}_{p^n} = \mathbf{k}[X]/(h(X)).$$

Since the degree n of h is not a multiple of p, the polynomial h is separable and

$$\mathbb{F}_{p^n}[T]/(h(T))$$

has no nonzero nilpotent elements. Then α^n is nonzero thus $c \in \mathbb{F}_p(\zeta)$ is also nonzero. It follows that α is invertible, indeed

$$\alpha^{-1} = (1 \otimes c^{-1})\alpha^{n-1}.$$

□

Definition 5.2.9 (Kummer constant)**.** Let $\alpha \in A_n$ be a nonzero solution of (H90) for the root ζ. The constant $c \in \mathbb{F}_p(\zeta)$ such that

$$\alpha^n = 1 \otimes c$$

is called the *Kummer constant* of α.

One key property of the solutions of (H90) is still missing, and we need a new notation to express it. Let K and L be two finite field extensions of $\mathbf{k}$ and let $\mu \in L$ be an element of degree d over $\mathbf{k}$. As used several times already, we know that an element

$$\beta \in K \otimes \mathbf{k}(\mu) \subset K \otimes L$$

can be uniquely written as

$$\beta = \sum_{j=0}^{d-1} x_j \otimes \mu^j,$$

and we set

$$\lfloor \beta \rfloor_\mu = x_0.$$

Proposition 5.2.10 ([All02a, Proposition 3.6]). *Let α be a nonzero solution of (H90) for the root ζ, then*

$$\lfloor \alpha \rfloor_\zeta$$

is a generating element of the extension

$$\mathbb{F}_{p^n}/\mathbf{k}.$$

Proof. Let $r = [\mathbb{F}_p(\zeta) : \mathbf{k}]$ and

$$P = X^r - \sum_{j=0}^{r-1} z_j X^j$$

the minimal polynomial of ζ over $\mathbf{k}$. Let also

$$\alpha = \sum_{j=0}^{r-1} a_j \otimes \zeta^j.$$

It follows that

$$\begin{aligned}
\sum_{j=0}^{r-1} \sigma(a_j) \otimes \zeta^j &= (\sigma \otimes 1)(\alpha) \\
&= (1 \otimes \zeta)\alpha \\
&= \sum_{j=0}^{r-1} a_j \otimes \zeta^{j+1} \\
&= \sum_{j=0}^{r-2} a_j \otimes \zeta^{j+1} + a_{r-1} \otimes (\sum_{i=0}^{r-1} z_i \zeta^i) \\
&= a_{r-1} z_0 \otimes 1 + \sum_{j=1}^{r-1} (a_{j-1} + a_{r-1} z_j) \otimes \zeta^j,
\end{aligned}$$

we thus have

$$\begin{cases} \sigma(a_0) = a_{r-1} z_0 \\ \sigma(a_j) = a_{j-1} + a_{r-1} z_j \text{ for all } 1 \leq j \leq r-1. \end{cases}$$

With these equations, we will prove that

$$\mathbb{F}_p(a_0) = \mathbb{F}_{p^n},$$

thus concluding the proof. We have $\sigma(a_0) \in \mathbb{F}_p(a_0)$ and $z_0 \in \mathbf{k}$. Since z_0 is nonzero, we have that

$$a_{r-1} = \sigma(a_0) z_0^{-1} \in \mathbb{F}_p(a_0).$$

Going from $j = r-1$ down to 1, it also follows that $a_{j-1} \in \mathbb{F}_p(a_0)$. Now assume that $e \in \mathbb{N}$ is an integer such that

$$\sigma^e(x) = x$$

for all $x \in \mathbb{F}_p(a_0)$, and such that $e < n$, *i.e.*

$$\mathbb{F}_p(a_0) \subsetneq \mathbb{F}_{p^n}.$$

Since all the a_j are in $\mathbb{F}_p(a_0)$, we also have

$$\begin{aligned}(1 \otimes \zeta^e)\alpha &= (\sigma \otimes 1)^e(\alpha) \\ &= (\sigma^e \otimes 1)(\alpha) \\ &= \alpha\end{aligned}$$

and thus $\zeta^e = 1$, which is a contradiction since ζ is a primitive n-th root of unity and $e < n$. Therefore we must have $e = n$ and $\mathbb{F}_p(a_0) = \mathbb{F}_{p^n}$. □

Remark 5.2.11. With the equations

$$\begin{cases} \sigma(a_0) = a_{r-1}z_0 \\ \sigma(a_j) = a_{j-1} + a_{r-1}z_j \text{ for all } 1 \leq j \leq r-1 \end{cases}$$

found in the proof of Proposition 5.2.10, we note that we can in fact also recover α from $\lfloor \alpha \rfloor_\zeta$.

Similarly to the case where $n \mid p-1$, where the solutions $\alpha \in \mathbb{F}_{p^n}$ of (H90) directly generate $\mathbb{F}_{p^n}$, we now know that the solutions $\alpha \in A_n$ of (H90) in the general case can also be used to generate $\mathbb{F}_{p^n}$ through $\lfloor \alpha \rfloor_\zeta$. We can study these solutions a bit further: in fact the element $\lfloor \alpha \rfloor_\zeta$ essentially depends on α^n, rather than on α.

Proposition 5.2.12. *Let*

$$\alpha = \sum_{j=0}^{r-1} a_j \otimes \zeta^j \in A_n$$

be a nonzero solution of (H90) *for* ζ *and let* $\alpha^n = 1 \otimes c$. *There are exactly* n *elements* $x \in A_n$ *that are solutions of* (H90) *for* ζ *and such that*

$$x^n = 1 \otimes c.$$

These elements are the

$$(1 \otimes \zeta^u)\alpha = (\sigma^u \otimes 1)(\alpha)$$

for $0 \leq u \leq n-1$. *The corresponding generating elements of* $\mathbb{F}_{p^n}/\mathbf{k}$ *are the*

$$\lfloor (\sigma^u \otimes 1)(\alpha) \rfloor_\zeta = \sigma^u(a_0);$$

they all have the same minimal polynomial, which is an irreducible polynomial defining $\mathbb{F}_{p^n}/\mathbf{k}$ *that only depends on the Kummer constant* $c \in \mathbb{F}_p(\zeta)$.

Proof. The solutions of (H90) form a one-dimensional $1 \otimes \mathbb{F}_p(\zeta)$-vector space, so all solutions are of the form

$$x = (1 \otimes \lambda)\alpha$$

with $\lambda \in \mathbb{F}_p(\zeta)$. Since we also ask for $x^n = \alpha^n$, we must have $\lambda^n = 1$ and thus

$$\lambda = \zeta^u$$

for some $0 \leq u \leq r-1$. Conversely all the solutions of the form $x = (1 \otimes \zeta^u)$ verify $x^n = \alpha^n$. Therefore, if x is such an element, we have

$$\begin{aligned}x &= (\sigma^u \otimes 1)(\alpha) \\ &= \sum_{j=0}^{r-1} \sigma^u(a_j) \otimes \zeta^j,\end{aligned}$$

thus

$$\lfloor x \rfloor_\zeta = \sigma^u(a_0).$$

All these elements are conjugates, thus share the same minimal polynomial, that is known to define $\mathbb{F}_{p^n}/\mathbf{k}$ by Proposition 5.2.10 and that depends only on the constant $c \in \mathbb{F}_p(\zeta)$. □

5.2.3 The isomorphism algorithm

Let $p \in \mathbb{N}$ a prime number and $n \in \mathbb{N}$ be an integer such that $\gcd(n, p) = 1$. Let K and L be two finite fields of cardinality p^n, such that

$$K \cong L \cong \mathbb{F}_{p^n}.$$

Let ζ be a primitive n-th root of unity. In this section, we show how to construct a single isomorphism between K and L, using the results of Section 5.2.2. How to construct embeddings between extensions of different degrees, and how to deal with a lattice of embeddings, is discussed in Chapter 7, together with additionnal results on Kummer algebras. We let

$$A_K = K \otimes \mathbb{F}_p(\zeta)$$

and

$$A_L = L \otimes \mathbb{F}_p(\zeta)$$

be the two Kummer algebras constructed with K and L and with the same field of scalars $\mathbb{F}_p(\zeta)$. The next proposition is the generalization of Proposition 5.2.2.

Proposition 5.2.13. *Let $\alpha_K \in A_K$ (respectively $\alpha_L \in A_L$) be a nonzero solution of (H90) for the root ζ in the Kummer algebra A_K (resp. A_L) and let $c_K \in \mathbb{F}_p(\zeta)$ (resp. $c_L \in \mathbb{F}_p(\zeta)$) the Kummer constant of α_K (resp. α_L). The quotient*

$$c_K/c_L$$

is a n-th power in $\mathbb{F}_p(\zeta)$, and if

$$\kappa^n = c_K/c_L$$

then the map sending $\lfloor \alpha_K \rfloor_\zeta$ to $\lfloor (1 \otimes \kappa)\alpha_L \rfloor_\zeta$ is an isomorphism from K to L.

Proof. The proof is very similar to the one of Proposition 5.2.2. Let

$$\phi : K \to L$$

be an isomorphism from K to L. Then $\phi \otimes 1$ is a morphism of algebras from A_K to A_L and

$$(\phi \otimes 1)(\alpha_K)$$

is a solution of (H90) for ζ in A_L. Therefore there exist $\lambda \in \mathbb{F}_p(\zeta)$ such that

$$(\phi \otimes 1)(\alpha_K) = \lambda \alpha_L.$$

The Kummer constant of $(\phi \otimes 1)(\alpha_K)$ is still c_K, and we have

$$c_K/c_L = \lambda^n.$$

Hence, if κ is an n-th root of c_K/c_L, we have

$$((1 \otimes \kappa)\alpha_L)^n = c_K,$$

i.e. the elements α_K and $(1 \otimes \kappa)\alpha_L$ share the same Kummer constant. By Proposition 5.2.12, the elements $\lfloor \alpha_K \rfloor_\zeta$ and $\lfloor (1 \otimes \kappa)\alpha_L \rfloor_\zeta$ have the same minimal polynomial and thus describe an isomorphism from K to L by Proposition 5.1.1. □

Finally, Lenstra-Allombert algorithm, in the general case, consists in

1. finding $\alpha_K \in K \otimes \mathbb{F}_p(\zeta)$ such that
$$(\sigma \otimes 1)(\alpha_K) = (1 \otimes \zeta)\alpha_K$$
and $\alpha_L \in L \otimes \mathbb{F}_p(\zeta)$ such that
$$(\sigma \otimes 1)(\alpha_L) = (1 \otimes \zeta)\alpha_L;$$
2. computing an n-th root $\kappa \in \mathbb{F}_p(\zeta)$ of the quotient α_K^n/α_L^n;
3. returning the isomorphism described by $\lfloor \alpha_K \rfloor_\zeta \mapsto \lfloor (1 \otimes \kappa)\alpha_L \rfloor_\zeta$.

The computational cost of Lenstra-Allombert algorithm resides in the computation of (H90) solutions, *i.e.* Step 1 of the previous list.

5.2.4 Computing (H90) solutions

In this section, we briefly present the different known solutions to compute (H90) solutions, details can be found in [BDFD+17]. Allombert first proposed to use linear algebra. The idea is to compute the matrix M of the Frobenius automorphism σ of $\mathbb{F}_{p^n}/\mathbf{k}$, that is the same as the matrix of $\sigma \otimes 1$, and then to compute an eigenvector of M over
$$1 \otimes \mathbb{F}_p(\zeta) \cong \mathbb{F}_p(\zeta)$$
for the eigenvalue
$$1 \otimes \zeta \cong \zeta.$$
Allombert later revised his algorithm [All02a, All02b]: instead of directly using linear algebra over $\mathbb{F}_p(\zeta)$, we can use the factorization
$$P(X) = (X - \zeta)b(X)$$
where P is the minimal polynomial of ζ over $\mathbf{k}$. If we write
$$P = X^r + \sum_{j=0}^{r-1} z_j X^j,$$
we can check that
$$b(X) = \sum_{j=0}^{r-1} b_j(X)\zeta^j, \text{ where } \begin{cases} b_{r-1}(X) = 1, \\ b_{j-1}(X) = b_j(X)X + z_j \text{ for all } 0 \leq j \leq r-1. \end{cases}$$
Indeed, direct computation shows that
$$(X - \zeta)b(X) = b_0(X)X + z_0 = b_{-1}(X)$$
and Horner 's rule also shows that
$$b_{-1}(X) = P(X).$$

We then get a solution of (H90) by evaluating $b(X)$ at an element in the kernel of

$$P(\sigma) = (\sigma - \zeta \operatorname{Id}) \circ b(\sigma),$$

hence we still use linear algebra, but over $\mathbf{k} = \mathbb{F}_p$ instead of $\mathbb{F}_p(\zeta)$ this time. A different strategy follows from the fact that if $x \in \mathbb{F}_{p^n}$, then

$$\alpha_x = \sum_{j=0}^{n-1} \sigma^j(x) \otimes \zeta^{-j-1}$$

verifies

$$(\sigma \otimes 1)(\alpha_x) = (1 \otimes \zeta)\alpha_x.$$

This is also a consequence of the factorization

$$\begin{aligned} X^n - 1 &= (X - \zeta) \sum_{j=0}^{n-1} \zeta^{-j-1} X^j \\ &= (X - \zeta)\Theta(X). \end{aligned}$$

The question is now whether the solution α_x is nonzero. By the theorem on character independence [Lan04, Chapter VI, §4] the maps $1, \sigma, \sigma^2, \ldots, \sigma^{n-1}$, all distinct, are independent. Therefore, the $\mathbf{k}$-linear map

$$x \mapsto \alpha_x$$

cannot be indentically zero on $\mathbb{F}_{p^n}$ and has rank at least 1. A random α_x thus has a probability of being zero less than $1/p$, so we need $O(1)$ trials to find a nonzero α_x at random. Depending on the relative size of p, n, and $s = [\mathbb{F}_p(\zeta) : \mathbf{k}]$, the computational cost of finding a nonzero α_x is discussed in details in [BDFD+17] and is bounded by $O(M(n^2)\log(n) + M(n)\log(p))$. If $\omega = 3$, where ω is the exponent of matrix multiplication, then the cost is at best quadratic in n.

5.3 The embedding evaluation problem

Let us first recall the problem. Let $\mathbf{k} = \mathbb{F}_p$ the prime field with p elements, where $p \in \mathbb{N}$ is a prime number, and let K and L be two finite extensions of $\mathbf{k}$. Let $m = [K : \mathbf{k}]$ and $n = [L : \mathbf{k}]$ be the respective degrees of the extensions $K/\mathbf{k}$ and $L/\mathbf{k}$ and assume that

$$m \mid n.$$

Let $\alpha \in K$ and $\beta \in L$ be two elements such that the $\mathbf{k}$-linear map

$$\phi : \alpha \mapsto \beta$$

sending α to β describes an embedding from K to L. The *embedding evaluation problem* consists in three sub-goals: given α and β defined above and elements $\gamma \in K$, $\delta \in L$:

- compute $\phi(\gamma) \in L$;
- test if $\delta \in \phi(K)$;
- if $\delta \in \phi(K)$, then compute $\phi^{-1}(\delta) \in K$.

In this section too, we follow the presentation of [BDFD+17] and we develop three solutions constructed on top of each other.

5.3.1 Linear algebra

Until the end of the section, we assume that the elements in K are represented on the monomial basis

$$(1, X, \ldots, X^{m-1})$$

and the elements in L are represented on the monomial basis

$$(1, Y, \ldots, Y^{n-1}).$$

We are particularly interested in the $\mathbf{k}$-vector space structure of K and L and in order to emphasize which basis is used, we let

$$V_X$$

be the vector space K equipped with the basis $(1, X, \ldots, X^{m-1})$ and

$$V_Y$$

be the vector space L equipped with the basis $(1, Y, \ldots, Y^{n-1})$. Similarly, we let

$$V_\alpha$$

be the vector space K equipped with the basis $(1, \alpha, \ldots, \alpha^{m-1})$ and

$$V_\beta$$

be the subspace $V_\beta \subset L$ equipped with the basis $(1, \beta, \ldots, \beta^{m-1})$, that is isomorphic to V_α. Since the map ϕ is $\mathbf{k}$-linear, we can store the $n \times m$ matrix representing ϕ in the monomial bases and evaluate ϕ via a matrix-vector product. We know that ϕ maps α to β, hence we write ϕ as the composition of three maps

$$V_X \xrightarrow{\sim} V_\alpha \xrightarrow{\sim} V_\beta \hookrightarrow V_Y.$$

First, we apply a change of basis from V_X to V_α. Then we apply the isomorphism from V_α to V_β, that is represented by the identity matrix. Finally we write the obtained element, expressed in $(1, \beta, \ldots, \beta^{m-1})$, in the monomial basis of L, *i.e.* we embed V_β in V_Y. The map $V_\beta \hookrightarrow V_Y$ is represented by an $n \times m$ matrix whose columns are the vectors $(\beta^j)_{0 \leq j \leq m-1}$ expressed in the monomial basis of L. The inverse of this map is obtained by solving a linear system. Similarly, the map $V_X \xrightarrow{\sim} V_\alpha$ is represented by the $m \times m$ matrix whose columns are the vectors $(X^j)_{0 \leq j \leq m-1}$ expressed in the basis $(1, \alpha, \ldots, \alpha^{m-1})$, that is also the inverse of the matrix whose columns are the vectors $(\alpha^j)_{0 \leq j \leq m-1}$ expressed in the monomial basis of K. The cost of the evaluation of ϕ and its inverse is thus dominated by the solving of the linear system, that is $O(m^{\omega-1}n)$. This cost can be reduced to $O(mn)$ with some precomputation, *e.g.* an LU decomposition, but the biggest drawback of the linear algebra approach is the memory complexity rather than the time complexity. Indeed, storing the matrices requires $O(mn)$ elements in $\mathbf{k}$.

5.3.2 Inverse maps and duality

The first improvement to the linear algebra method consists in replacing the linear system solving used in the computation of the inverse of

$$V_\beta \hookrightarrow V_Y$$

by simpler operations, such as matrix-vector product, in order to reduce the overall complexity. We first recall facts about bilinear forms and duality, the presentation follows the one in [BDFD+17] and [DFDS14], and a standard presentation is also available in [Lan04]. Recall that

$$K \cong \mathbb{F}_{p^m}$$

admits a monomial basis $(1, X, \ldots, X^{m-1})$ and that

$$L \cong \mathbb{F}_{p^n}$$

admits a monomial basis $(1, Y, \ldots, Y^{n-1})$. The trace $\mathrm{Tr}_{K/\mathbf{k}}$ from K to $\mathbf{k}$ defines a non-degenerate bilinear form denoted by

$$\langle x, y \rangle_K = \mathrm{Tr}_{K/\mathbf{k}}(xy).$$

Therefore, we can define a *dual basis* $(X_0^*, X_1^*, \ldots, X_{m-1}^*)$ to $(1, X, \ldots, X^{m-1})$, characterized by

$$\left\langle X^j, X_i^* \right\rangle = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ otherwise.} \end{cases}$$

Similarly, we can define a non-degenerate bilinear form from L to $\mathbf{k}$ by

$$\langle x, y \rangle_L = \mathrm{Tr}_{L/\mathbf{k}}(xy),$$

and a dual basis $(Y_0^*, \ldots, Y_{n-1}^*)$ to $(1, \ldots, Y^{n-1})$ corresponding to this bilinear form. Now, let

$$\psi : K \hookrightarrow L$$

be an embedding. It is a $\mathbf{k}$-linear map and thus there exists a unique *dual map* ψ^t, such that

$$\langle \psi(x), y \rangle_L = \left\langle x, \psi^t(y) \right\rangle_K$$

for all $x \in K$ and $y \in L$. Furthermore, if M is the matrix representing ψ in the monomial bases of K and L, then its transposed matrix M^t represents the map ψ^t in the *dual bases*. This allows us to efficiently compute ψ^t, because the conversions from the monomial basis to the dual basis of $\mathbb{F}_{p^m}$ can be done at a cost of $O(M(m)\log(m))$ operations in $\mathbf{k}$, as explained in [DFDS14]. Fortunately, the map ψ^t, that is easy to compute, is closely related to ψ^{-1}, the map that we want. In particular, if $K \cong L$ and ψ is an isomorphism of fields, we have that ψ^t is exactly ψ^{-1}. Indeed, in that case, the map ψ preserves the bilinear form, *i.e.* for all $x, y \in K$, we have

$$\langle \psi(x), \psi(y) \rangle_L = \langle x, y \rangle_K .$$

We thus have

$$\left\langle x, (\psi^t \circ \psi)(y) \right\rangle_K = \langle x, y \rangle_K ,$$

and it follows that $\psi^t \circ \psi$ is the identity map, because the bilinear form is non-degenerate. Now if K and L are not isomorphic, *i.e.* $m < n$, we can still use ψ^t to recover ψ^{-1}. Let

$$d = [L : K] = \frac{n}{m}$$

and

$$x \in \psi(K) \subsetneq L,$$

we have

$$\begin{aligned}\mathrm{Tr}_{L/\mathbf{k}}(x) &= \sum_{i=0}^{n-1} \sigma^i(x) \\ &= \sum_{i=0}^{d-1} \sum_{j=0}^{m-1} \sigma^{im+j}(x) \\ &= \sum_{i=0}^{d-1} \sum_{j=0}^{m-1} \sigma^{j}(x) \\ &= \sum_{i=0}^{d-1} \mathrm{Tr}_{\psi(K)/\mathbf{k}}(x)\end{aligned}$$

and thus it follows that

$$\mathrm{Tr}_{L/\mathbf{k}}(x) = d\,\mathrm{Tr}_{\psi(K)/\mathbf{k}}(x).$$

Hence, in that case, the map ψ does not preserve the bilinear form, but we still have

$$\langle \psi(x), \psi(y) \rangle_L = d\,\langle x, y \rangle_K$$

for all $x, y \in K$. If d is not a multiple of the characteristic p, then the map $d^{-1}\psi^t$ is the inverse of ψ, by the same argument as before. Otherwise, let $x \in \psi(K) \subsetneq L$ and let $u \in L$ an element such that

$$\mathrm{Tr}_{L/\psi(K)}(u) = 1.$$

Then, by transitivity of the trace, and by $\psi(K)$-linearity of the trace $\mathrm{Tr}_{L/\psi(K)}$, it follows that

$$\begin{aligned}\mathrm{Tr}_{L/\mathbf{k}}(ux) &= \mathrm{Tr}_{\psi(K)/\mathbf{k}}(\mathrm{Tr}_{L/\psi(K)}(ux)) \\ &= \mathrm{Tr}_{\psi(K)/\mathbf{k}}(x\,\mathrm{Tr}_{L/\psi(K)}(u)) \\ &= \mathrm{Tr}_{\psi(K)/\mathbf{k}}(x).\end{aligned}$$

Therefore we have

$$\begin{aligned}\langle x, y \rangle_K &= \mathrm{Tr}_{K/\mathbf{k}}(xy) \\ &= \mathrm{Tr}_{\psi(K)/\mathbf{k}}(\psi(x)\psi(y)) \\ &= \mathrm{Tr}_{L/\mathbf{k}}(u\psi(x)\psi(y)) \\ &= \langle \psi(x), u\psi(y) \rangle_L\,,\end{aligned}$$

and if we let U the map defined by $z \mapsto uz$, we conclude that

$$\psi^t \circ U \circ \psi = \psi^t \circ U^t \circ \psi$$

is the identity map, and we can once again compute ψ^{-1}. Let us go back to our original problem: we want to compute the embedding

$$\phi : K \to L$$

that is described by $\phi(\alpha) = \beta$. Recall that we can decompose ϕ into three maps

$$V_X \xrightarrow{\sim} V_\alpha \xrightarrow{\sim} V_\beta \hookrightarrow V_Y,$$

and that we want to be able to compute both ϕ and its inverse ϕ^{-1}. The map $V_\alpha \xrightarrow{\sim} V_\beta$ is represented by the identity matrix, thus it is straighforward to compute. The map $V_\alpha \xrightarrow{\sim} V_X$ (the inverse of $V_X \xrightarrow{\sim} V_\alpha$) is represented, in the monomial bases of V_α and V_X, by the matrix whose columns are the vectors $(\alpha^j)_{0\leq j\leq m-1}$ in the monomial basis of K. The map $V_\beta \hookrightarrow V_Y$ is similarly represented by the vectors $(\beta^j)_{1\leq j\leq m-1}$ in the monomial basis of L. Both these maps are embeddings, thus we can compute their inverse using duality theory instead of solving linear systems like described in Section 5.3.1. The inverse of $V_\beta \hookrightarrow V_Y$ is evaluated by multiplying by a fixed element u that has a trace equal to 1, then the result is converted in the dual basis of V_Y, a matrix-vector product with the transposed matrix of $V_\beta \hookrightarrow V_Y$ is computed, and the product is then converted back to the monomial basis of V_Y. The inverse of the embedding $V_\alpha \xrightarrow{\sim} V_X$ is obtained similarly, but without the need to multiply by an element u, because the embedding is actually an isomorphism in that case.

It is possible to apply these steps on every element in V_Y, although ϕ^{-1} is defined only on the image of ϕ, *i.e.* the subfield

$$\phi(K) \subset L.$$

If the element we apply our steps to is not in $\phi(K)$, we just obtain an arbitrary projection in K. Indeed, if $x \in K$ and $y \in L$, we can decompose y as

$$\begin{aligned} y &= y - \mathrm{Tr}_{L/K}(uy) + \mathrm{Tr}_{L/K}(uy) \\ &= y' + \mathrm{Tr}_{L/K}(uy), \end{aligned}$$

where $u \in L$ is an element such that $\mathrm{Tr}_{L/K}(u) = 1$ and $y' = y - \mathrm{Tr}_{L/K}(uy)$. Then we have

$$\begin{aligned} \left\langle \phi(x), uy' \right\rangle_L &= \mathrm{Tr}_{L/\mathbf{k}}(\phi(x)uy') \\ &= \mathrm{Tr}_{K/\mathbf{k}}(\mathrm{Tr}_{L/K}(\phi(x)uy')) \\ &= \mathrm{Tr}_{K/\mathbf{k}}(\mathrm{Tr}_{L/K}(\phi(x)u(y - \mathrm{Tr}_{L/K}(uy)))) \\ &= \mathrm{Tr}_{K/\mathbf{k}}(\mathrm{Tr}_{L/K}(\phi(x)uy)) - \mathrm{Tr}_{K/\mathbf{k}}(\mathrm{Tr}_{L/K}(\phi(x)u\,\mathrm{Tr}_{L/K}(uy))) \\ &= \mathrm{Tr}_{K/\mathbf{k}}(x\,\mathrm{Tr}_{L/K}(uy)) - \mathrm{Tr}_{K/\mathbf{k}}(x\,\mathrm{Tr}_{L/K}(uy)\,\mathrm{Tr}_{L/K}(u)) \\ &= \mathrm{Tr}_{K/\mathbf{k}}(x\,\mathrm{Tr}_{L/K}(uy)) - \mathrm{Tr}_{K/\mathbf{k}}(x\,\mathrm{Tr}_{L/K}(uy)) \\ &= 0, \end{aligned}$$

therefore

$$\begin{aligned} \langle \phi(x), uy \rangle_L &= \left\langle \phi(x), uy' \right\rangle_L + \left\langle \phi(x), u\,\mathrm{Tr}_{L/K}(uy) \right\rangle_L \\ &= \left\langle \phi(x), u\,\mathrm{Tr}_{L/K}(uy) \right\rangle_L \\ &= \left\langle x, \mathrm{Tr}_{L/K}(uy) \right\rangle_K, \end{aligned}$$

that means that applying our solution on y results in the element $\mathrm{Tr}_{L/K}(uy)$ in V_β, which coincides with y if y is in the subfield $\phi(K) \subset L$. In order to test if an element y is in $\phi(K)$, the best way is then to project y to $z = \mathrm{Tr}_{L/K}(uy)$, and then check that

$$\phi(z) = y.$$

After precomputation of the matrices, the most expensive operation is the matrix-vector product, that costs $O(mn)$ operations in $\mathbf{k}$. It is thus better than the cost of solving a linear system. Nevertheless, the problem of the memory complexity remains.

5.3.3 Modular composition

In order to tackle the memory complexity problem, we can replace the matrix computations by modular compositions, a technique initiated by Shoup [Sho94, Sho95, Sho99]. Indeed, the computation of the embedding

$$V_\beta \hookrightarrow V_Y$$

is precisely a modular composition computation: we want to express a polynomial

$$\gamma = \sum_{j=0}^{m-1} c_j \beta^j,$$

representing an element in V_β, into the monomial basis $(1, \ldots, Y^{n-1})$, given the polynomial expression of β in V_Y

$$\beta = \sum_{j=0}^{n-1} b_j Y_j.$$

If g is the polynomial defining L over $\mathbf{k}$, *i.e.*

$$L \cong \mathbf{k}[Y]/(g(Y)),$$

then the computation of $V_\beta \hookrightarrow V_Y$ is exactly the modular composition

$$\gamma(\beta(Y)) \mod g(Y),$$

and this can be done efficiently by a dedicated algorithm [KU08]. In order to compute the inverse of $V_\beta \hookrightarrow V_Y$, we cannot use the same algorithm, since this is not a modular composition problem, but we use a generalization of the duality results of Section 5.3.2 called the *transposition principle*. This technique, also known as *Tellengen's principle* [BLS03, DF10, DFS10], allows one to *transpose* an algorithm used to compute a linear map into a new algorithm that computes the transposed linear map, without changing the complexity of the algorithm. We can use the transposition principle with the modular composition, since the map

$$\gamma \to \gamma(\beta) \mod g$$

is linear. The dual problem to modular composition was called *power projection* by Shoup, who popularized the transposition principle. It takes as inputs the polynomials $\beta, g \in \mathbf{k}[Y]$, and an element $\gamma^\vee$ in $\mathbf{k}[Y]^\vee$, the dual space of $\mathbf{k}[Y]$, *i.e.* the space of linear forms on $\mathbf{k}[Y]$. Its output is the list of elements

$$\gamma^\vee(1), \gamma^\vee(\beta), \gamma^\vee(\beta^2), \ldots, \gamma^\vee(\beta^{n-1})$$

in $\mathbf{k}$. Thanks to the transposition, the power projection problem can be solved within the same complexity bound as modular composition. Finally, the inverse of the embedding $\phi : V_\beta \hookrightarrow V_Y$ is computed by Algorithm 7. This algorithm takes $y \in L$ and computes $\mathrm{Tr}_{L/K}(uy)$, where u is still an element of relative trace equal to one:

$$\mathrm{Tr}_{L/K}(u) = 1.$$

As discussed in Section 5.3.2, this is also the result of applying

$$\phi^t \circ U = \phi^{-1}$$

Algorithm 7 Inverse embedding

Input: An element $y \in L$, and two precomputed values: an element $\beta \in L$ generating a subfield isomorphic to K and an element $u \in L$ such that $\mathrm{Tr}_{L/K}(u) = 1$.

Output: The element $\mathrm{Tr}_{L/K}(uy)$ written in the basis $(1, \beta, \ldots, \beta^{m-1})$.

1: Compute the minimal polynomial of β over $\mathbf{k}$;
2: compute $y' = uy$;
3: convert y' to the dual basis $(Y_0^*, \ldots, Y_{n-1}^*)$;
4: compute $z = \mathrm{Tr}_{L/K}(y')$ using *power projections*;
5: convert z to the monomial basis $(1, \beta, \ldots, \beta^{m-1})$;
6: **return** z.

to y, where $U : y \to uy$ is the multiplication-by-u map. When y is in the subfield $\phi(K) \subsetneq L$, we obtain the element

$$\mathrm{Tr}_{L/K}(uy) = y$$

expressed in the basis $(1, \ldots, \beta^{m-1})$. Since the trace is computed using power projection, rather than by a transposed matrix-vector product, this solves the quadratic memory complexity issue. The minimal polynomial of β, computed in Line 1, is required to perform conversions between the monomial and the dual basis generated by β. It is computed with $O(M(n)\log(n))$ operations, using the Berlekamp-Massey algorithm. Conversions are then also computed with $O(M(n)\log(n))$ operations with the algorithms in [DFDS14]. Finally, the power projection costs $O(n^{(\omega+1)/2})$ operations with one of the algorithms in [Sho95, KU08], thus the total complexity of Algorithm 7 is $O(n^{(\omega+1)/2})$. We have yet to see how to compute the element $u \in L$. If

$$d = [L : K]$$

is not divisible by the characteristic p of $\mathbf{k}$, then d^{-1} is such an element. Otherwise we can take any element whose trace is nonzero and divide it by its trace to have an element whose trace is exactly equal to 1. To obtain an element whose trace is nonzero, we take random elements and compute their trace: a number of $O(1)$ trials is expected until a suitable element is found. Computing one trace can be done using $O(n^{(\omega+1)/2}\log(n) + M(n)\log(p))$ operations, thus the computation of u has a cost of $O(n^{(\omega+1)/2}\log(n) + M(n)\log(p))$. To conclude, after this precomputation, all the sub-problems of the *embedding evaluation problem* can be solved using $O(n^{(\omega+1)/2})$ operations in $\mathbf{k}$.

Chapter 6

From a single finite field to plenty: lattice of embeddings

In Chapter 5, we have seen algorithms to comput. isomorphisms, or embeddings, between pairs of finite fields. Now, we want to integrate these algorithms in a larger global system with potentially as many finite fields as we want.

Contents

6.1 The compatibility problem 111
6.2 Conway polynomials 112
6.3 The Bosma-Canon-Steel framework 114
6.3.1 The Bosma-Canon-Steel algorithm 114
6.3.2 Implementation in Nemo 120

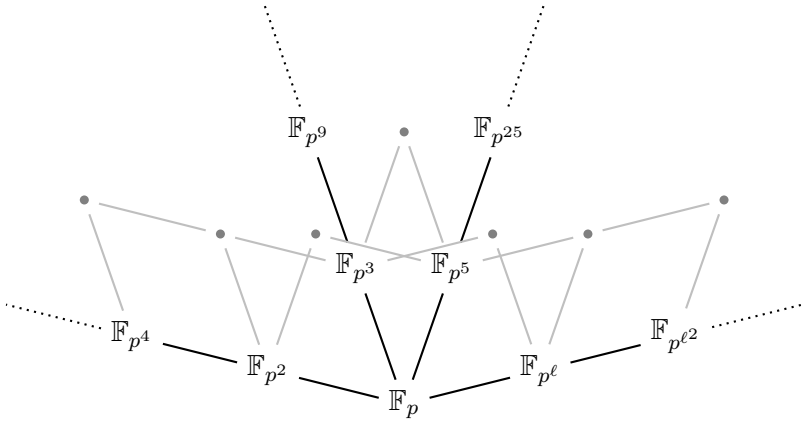


Figure 6.1: Extensions of $\mathbb{F}_p$.

6.1 The compatibility problem

Now that we know how to go from one finite field $\mathbb{F}_{p^a}$ to another $\mathbb{F}_{p^b}$, with $p \in \mathbb{N}$ a prime number and

$$a \,|\, b$$

two integers, we would like to manage more than two finite fields simultaneously. In other words, given a family

$$\mathcal{F} = \{\mathbb{F}_{p^a} \,|\, a \in E\}$$

for E a subset of $\mathbb{N} \setminus \{0\}$, we want to be able to compute an embedding

$$\mathbb{F}_{p^a} \hookrightarrow \mathbb{F}_{p^b}$$

each time that we have $\mathbb{F}_{p^a}, \mathbb{F}_{p^b} \in \mathcal{F}$ with a dividing b, while maintaining *compatibility* between all these embeddings. From a pratical point of view, we want to build a computer algebra system where the users can embed the field they are working with in a bigger finite field, or conversely if an element is known to belong to a smaller field than the ambient field, project it to a smaller field. In cryptology and coding theory, finite fields are ubiquitus, and some algorithms require frequent change of field. For example, in the quasi-polynomial algorithm for discrete logarithm in small characteristic by Granger, Kleinjung and Zumbrägel [GKZ14], we have to work with a tower of finite field extensions, and thus a computer algebra system automatically dealing with the changes would be very convenient in order to implement their algorithm. From a theoretical point of view, this leads to the question of the arithmetic in the algebraic closure of some finite field $\mathbb{F}_p$

$$\bar{\mathbb{F}}_p = \bigcup_{j \in \mathbb{N}\setminus\{0\}} \mathbb{F}_{p^j}$$

and its representation on a computer, a question that was for example investigated in [DFDS14].

If we *just* want to compute embeddings between two finite fields in $\mathcal{F}$ when it makes sense, we can use one of the algorithms presented in Chapter 5 each time we need it. In fact, we want to build a data structure Λ to represent all the extensions of $\mathbb{F}_p$ that we need, and additional sub-goals might be desirable.

Effective embeddings: for any pair of extensions $k \subset K$ in Λ, there exists an efficiently computable embedding $\phi : k \to K$, and algorithms to evaluate ϕ on k, and the section ϕ^{-1} on K.

Compatibility: the embeddings are *compatible*, *i.e.* for any triple $k \subset K \subset L$ in Λ, and embeddings $\phi : k \to K$, $\psi : K \to L$, $\chi : k \to L$ such as shown in Figure 6.2, one has $\chi = \psi \circ \phi$.

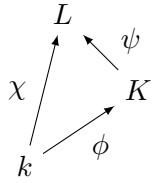


Figure 6.2: Embeddings between finite fields $k \subset K \subset L$.

Incrementality: the data associated with an extension (*e.g.* its irreducible polynomial, change-of-basis matrices, ...) must be computable efficiently and *incrementally*, *i.e.* adding a new field extension to Λ does not require recomputing data for all extensions already in Λ.

Uniqueness: any extension of $\mathbb{F}_p$ is determined by an irreducible polynomial whose definition only depends on the characteristic p and the degree of the extension.

Generality: extensions of $\mathbb{F}_p$ can be represented by arbitrary irreducible polynomials.

We cannot fulfill all these conditions simultaneously, as uniqueness and generality are in conflict with each other. One might prefer to look for uniqueness or generality, depending on the situation, but it would always be better to have effective embeddings, compatibility and incrementality. In order to achieve the "effective embeddings" condition, we can use the efficient algorithms presented in Chapter 5. The only problem is that if we have three finite fields

$$k \subset K \subset L$$

and three embeddings $\phi : k \to K$, $\psi : K \to L$, $\chi : k \to L$, there is no guaranty that the diagram of Figure 6.2 commutes, *i.e.*

$$\chi = \psi \circ \phi.$$

If we want to achieve compatibility, we have to provide extra-work. The two constructions presented in Sections 6.2 and 6.3 are both solutions oriented towards this goal. Conway polynomials yield *uniqueness* while the Bosma, Cannon and Steel framework grants *generality*.

6.2 Conway polynomials

Conway polynomials [Par, Sch92] are named after J. H. Conway, they were introduced by Parker in 1990 and provide embeddings that are easy to compute.

Definition 6.2.1 (Conway polynomials)**.** Let $p \in \mathbb{N}$ be a prime number. The m-th *Conway polynomial*

$$C_m \in \mathbb{F}_p[X]$$

is the lexicographically smallest monic irreducible polynomial of degree m that is also primitive (*i.e.* its roots generate $\mathbb{F}_{p^m}^\times$) and norm-compatible, *i.e.* if

$$m \,|\, n$$

we have

$$C_m(X^{\frac{p^n-1}{p^m-1}}) \mod C_n = 0.$$

The norm compatibility means that if one takes the norm relatively to the extension

$$\mathbb{F}_{p^n}/\mathbb{F}_{p^m}$$

of a root of C_n, one obtains a root of C_m. This compatibility will allow us to define compatible embeddings. Let $p \in \mathbb{N}$ be a prime number, $m, n \in \mathbb{N}$ be two integers such that

$$m \,|\, n$$

and let $C_m \in \mathbb{F}_p[X], C_n \in \mathbb{F}_p[Y]$ be the m-th and the n-th Conway polynomial. Let

$$k = \mathbb{F}_{p^m} = \mathbb{F}_p(\alpha_m) \cong \mathbb{F}_p[X]/(C_m)$$

and

$$K = \mathbb{F}_{p^n} = \mathbb{F}_p(\alpha_n) \cong \mathbb{F}_p[Y]/(C_n)$$

with $\alpha_m = \bar{X} \in k$ and $\alpha_n = \bar{Y} \in K$. Now, we define the embedding

$$\begin{array}{rccc} \phi_{k \hookrightarrow K} : & k & \hookrightarrow & K \\ & \alpha_m & \mapsto & (\alpha_n)^{\frac{p^n-1}{p^m-1}} \end{array}$$

that sends α_m to $(\alpha_n)^{\frac{p^n-1}{p^m-1}}$, the norm of α_n relative to k. Because this is a field morphism, it is implied that every polynomial in α_m is sent to the same polynomial in $(\alpha_m)^{\frac{p^n-1}{p^m-1}}$. That is indeed an embedding because of the norm-compatibility condition in Conway polynomials. This definition also provides compatibility thanks to the transitivity of the norm: let

$$k \subset K \subset L$$

three finite fields defined using Conway polynomials, let

$$N_{K/k}, N_{L/K}, N_{L/k}$$

be the respective norms of the extensions

$$K/k, L/K, L/k$$

and

$$\phi_{k \hookrightarrow K}, \phi_{K \hookrightarrow L}, \phi_{k \hookrightarrow L}$$

the corresponding embeddings. Then, for any element $x \in L$, we have

$$N_{L/k}(x) = N_{K/k}(N_{L/K}(x)),$$

and it follows, using the fact that $\phi_{K \hookrightarrow L}$ and $N_{K/k}$ commute, that

$$\phi_{k \hookrightarrow L} = \phi_{K \hookrightarrow L} \circ \phi_{k \hookrightarrow K}.$$

Therefore, using Conway polynomials to define all finite field extensions allows compatibility *a priori*: adding a new extension in the lattice does not require any computation and the form of the embedding is already known. The only condition is to define each extension using the Conway polynomial of the corresponding degree. These polynomials always exist so this is possible, although the best known algorithm to compute Conway polynomials has exponential complexity [HL04] in the degree of the polynomial. Because the cost of computing Conway polynomials is prohibitive, they are often precomputed and tabulated up to a certain degree in many computer algebra systems. As a result, most computer algebra systems use Conway polynomial to represent finite fields of small degrees, and switch to other representations when Conway polynomials are no longer available. This strategy leads to efficient lattices of compatibly embedded finite fields, at least in small degrees, but at the cost of generality, because the finite fields extensions cannot be user-defined and high degree extensions cannot be included in the lattice. We see in Section 6.3 that those issues can be adressed by using another method defined by Bosma, Canon, and Steel [BCS97], but the price to compute each embedding is higher.

6.3 The Bosma-Canon-Steel framework

In 1987, Bosma, Canon and Steel proposed a new framework [BCS97] to deal with lattices of finite field extensions. These algorithms were initially implemented in the computer algebra system MAGMA [BCP97]. The framework allows the user to define finite fields with custom polynomials, and compute the correct embeddings on the fly, in a way that guarantees that at least one compatible embedding will always exist between two finite fields, when that makes sense. To the best of my knowledge, this framework was only used in MAGMA until very recently. We first implemented the framework using Nemo and Flint and presented our software in the symbolic computation conference ISSAC in 2018 [DFRR18]. In 2019, the code was also added to the official branch of Nemo, and Nemo can now deal with finite field extensions and compatibly embed them, in the case where the characteristic p of the fields fits in a machine word. There are no theretical obstacles preventing from implementing the algorithm in arbitrarily large characteristic, but some tools were lacking at the C level (*i.e.* in Flint).

6.3.1 The Bosma-Canon-Steel algorithm

Notations and first results. Let $p \in \mathbb{N}$ be a prime number and q a power of p. We present the results over the prime field $\mathbb{F}_p$, but they can be extended to $\mathbb{F}_q$ without any new difficulty. Given two finite fields k and K with cardinalities

$$\#k = p^m$$

and

$$\#K = p^n$$

such that m divides n, we know that k can be embedded in K. In other words, we know that K contains one subfield

$$k' = \left\{x \in K \,|\, x^{p^m} = x\right\} \subset K$$

with

$$\#k' = p^m.$$

There is exactly one subfield k' in K that has the same cardinality as k, but there exist m different ways to map k to k'. One way of seeing that is to say that an element $\alpha \in k$ generating k has a minimal polynomial π over $\mathbb{F}_p$ of degree m, and that α can be mapped to any one of the m roots of π in K. This choice sets the embedding because any element in k can be written as

$$\sum_{j=0}^{m-1} a_j \alpha^j,$$

with $a_j \in \mathbb{F}_p$. We write $k \hookrightarrow K$ if an explicit embedding has been chosen. Let

$$k \subset K \subset L$$

with $k \hookrightarrow K$ and $k \hookrightarrow L$, as in Figure 6.3. The foundations of the Bosma-Canon-Steel framework lay on the fact that we precisely know how many compatible embeddings from K to L we can find.

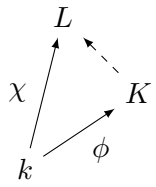


Figure 6.3: Incomplete embedding diagram.

Proposition 6.3.1 ([BCS97, Thm. 2.1]). *Let $l \mid m \mid n$ be three integers and let*

$$k \subset K \subset L$$

be three finite field extensions of $\mathbb{F}_p$, with respective cardinalities p^l, p^m and p^n. If k is embedded in K and L, i.e. *if $k \hookrightarrow K$ and $k \hookrightarrow L$, then there are m/l compatible embeddings from K to L.*

Proof. Let $\phi : k \to K$ be the embedding from k to K and $\chi : k \to L$ the embedding from k to L. Let $\widetilde{\psi} : K \to L$ be any embedding from K to L. The maps χ and $\widetilde{\psi} \circ \phi$ are two embeddings from k to L. There are only l such maps, because if α is a generating element in k and if π is its minimal polynomial over $\mathbb{F}_p$, then an embedding from k to L necessarily sends $\alpha \in k$ to a root $\beta \in L$ of π in L. The embedding is then uniquely defined by

$$\alpha \mapsto \beta$$

and there are l such roots, that are all conjugates, thus we know that there exists

$$\sigma \in \mathrm{Gal}(L/\mathbb{F}_p)$$

such that

$$\chi = \sigma \circ \widetilde{\psi} \circ \phi.$$

If we set

$$\psi = \sigma \circ \widetilde{\psi},$$

we obtain a compatible embedding $\psi : K \to L$ from K to L. If we compose ψ with a $\chi(k)$-automorphism of L, *i.e.* an automorphism of $\psi(K)$ that fixes the subfield $\chi(k)$, we still obtain a compatible embedding because the images of the elements in $\chi(k)$ are unchanged. Now, we know that

$$\#\,\mathrm{Gal}(\psi(K)/\chi(k)) = m/l,$$

thus we know that there are exactly m/l embeddings from K to L that are compatible with ϕ and χ. □

In the future, if we have two finite fields k and K such that $k \hookrightarrow K$ with an explicit embedding $\phi : k \to K$, we will often identify k with its isomorphic image $\phi(k) \cong k$ in K, for the sake of simplicity. We also have a constructive version of Proposition 6.3.1.

Proposition 6.3.2 ([BCS97, Thm. 2.2]). *Let $l \mid m \mid n$ be three integers and let*

$$k \subset K \subset L$$

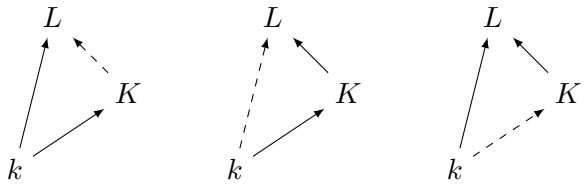


Figure 6.4: The different configurations with triangles.

be three finite field extensions of $\mathbb{F}_p$*, with respective cardinalities* p^l*,* p^m *and* p^n*. Assume that* $k \hookrightarrow K$ *and* $k \hookrightarrow L$*, and let* $\phi = \phi_{k\hookrightarrow K}$*,* $\chi = \phi_{k\hookrightarrow L}$ *be the respective embeddings. Let also* $\alpha \in K$ *be an element that generates* K *over* k *and*

$$\pi_k = \mathrm{Minpoly}_k(\alpha)$$

the minimum polynomial of α *over* k*. Let* $\rho \in L$ *be a root of* π_k *in* L*. Let* $\psi : K \to L$ *be the map defined by*

$$\psi\left(\sum_{j=0}^{m/l-1} x_j \alpha^j\right) = \sum_{j=0}^{m/l-1} x_j \rho^j,$$

using the unique representation of any element of K *as* $\sum_j x_j \alpha^j$ *with* $x_j \in k$*. Then* ψ *is an embedding from* K *to* L *that is compatible with* ϕ *and* χ*,* i.e. *we have*

$$\chi = \psi \circ \phi.$$

Proof. The root ρ in L exists since we know that π_k has a root in K and $K \subset L$. The map ψ is then well-defined because of the uniqueness of the representation of the elements in K as $\sum_j x_j \alpha^j$. The elements α and ρ have the same minimum polynomial so the map ψ is a homomorphism. The map ψ is also injective and thus it is an embedding from K to L. The compatibility condition holds by construction. □

Remark 6.3.3. In fact, in our implementation and even in Proposition 6.3.5, this result will not be used as is and we will usually take $\alpha \in K$ an element generating K over the base field $\mathbb{F}_p$ instead of over k. Indeed, if α generates K over $\mathbb{F}_p$, it also generates K over k. We can still take a root ρ of $\mathrm{Minpoly}_k(\alpha)$ because

$$\mathrm{Minpoly}_k(\alpha) \mid \mathrm{Minpoly}_{\mathbb{F}_p}(\alpha)$$

and so a root of $\mathrm{Minpoly}_k(\alpha)$ is necessarily a root of $\mathrm{Minpoly}_{\mathbb{F}_p}(\alpha)$. As a result, we can use the $\mathbb{F}_p$-algebra structure of K and still achieve compatibility with the subfield k.

Although the result of Proposition 6.3.1 captures only one configuration among the three different possibilities of Figure 6.4, it is the most important case. Indeed, if we already know $\phi_{k\hookrightarrow K}$ and $\phi_{K\hookrightarrow L}$ (the configuration in the middle of Figure 6.4), we can just set

$$\phi_{k\hookrightarrow L} = \phi_{K\hookrightarrow L} \circ \phi_{k\hookrightarrow K}.$$

Finally the configuration on the right, where we know $\phi_{k\hookrightarrow L}$ and $\phi_{K\hookrightarrow L}$, is also irrelevant because, by construction, this case never happens when working with the Bosma-Canon-Steel framework.

Description of the framework. The general strategy of Bosma, Canon, and Steel to compute and maintain the data of several finite fields and compatible embeddings between them is to ensure that some properties are conserved at all times. If $k = \mathbb{F}_{p^m}$ is a finite extension of $\mathbb{F}_p$, we define

$$\deg(k) = [k : \mathbb{F}_p] = m$$

the extension degree over $\mathbb{F}_p$.

Definition 6.3.4 (Lattice of compatibly embedded finite fields)**.** Let q be a power of p a prime number and $\mathbb{F}_p$ the finite field with q elements. The pair

$$\Lambda = (\mathcal{K}, \Phi),$$

where $\mathcal{K}$ is a collection of finite fields sharing the same base field $\mathbb{F}_p$ and Φ is a collection of embeddings between members of $\mathcal{K}$, is called a *lattice of compatibly embedded finite fields* if the following conditions are satisfied:

CE1 (unicity) for each pair (k, K) of elements in $\mathcal{K}$, there exists at most one corresponding embedding $\phi_{k\hookrightarrow K} \in \Phi$.

CE2 (reflexivity) For each $k \in \mathcal{K}$, the identity map $\mathrm{Id}_k = \phi_{k\hookrightarrow k}$ is in Φ.

CE3 (base subfield) There is exactly one $k \in \mathcal{K}$ such that $\deg(k) = 1$, and for all $K \in \mathcal{K}$, there exists $\phi_{\mathbb{F}_p\hookrightarrow K} \in \Phi$.

CE4 (invertibility) If $k \hookrightarrow K$ and $\deg(k) = \deg(K)$, then $K \hookrightarrow k$ and $\phi_{K\hookrightarrow k} = \phi_{k\hookrightarrow K}^{-1}$.

CE5 (transitivity) For any triple (k, K, L) of elements in $\mathcal{K}$, if $k \hookrightarrow K \hookrightarrow L$ then $k \hookrightarrow L$ and $\phi_{k\hookrightarrow L} = \phi_{K\hookrightarrow L} \circ \phi_{k\hookrightarrow K}$.

CE6 (intersections) For each $k, K, L \in \mathcal{K}$ such that $K \hookrightarrow L$ and $k \hookrightarrow L$, there exists $I \in \mathcal{K}$ such that $\deg(I) = \gcd(\deg(k), \deg(K))$ and $I \hookrightarrow k$, $I \hookrightarrow K$.

Conditions CE1 to CE5 are quite natural, some of them are technical and are not important in our implementation. For example, Condition CE3 is totally free in our implementation because we work with $q = p$ and the elements of our fields are represented by polynomials over $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$, thus the subfield $\mathbb{F}_p \subset \mathbb{F}_{p^m}$ is always represented by the constant polynomials and the embedding is trivial. Nevertheless, Condition CE6 is important both from a theoretical and on an implementation point of view: it ensures that the implicit embeddings between common subfields are made explicit, in order to prevent any possible future incompatibility, and it forces the computer algebra software to compute extra embeddings that the user might not need. Assume that we have embedded $\mathbb{F}_{p^4}$ and $\mathbb{F}_{p^6}$ into $\mathbb{F}_{p^{12}}$, there is now an implicit isomorphism between the two copies of the quadratic subfield $\mathbb{F}_{p^2}$ in $\mathbb{F}_{p^4}$ and $\mathbb{F}_{p^6}$. Any future embedding from $\mathbb{F}_{p^4}$ or $\mathbb{F}_{p^6}$ in an extension of $\mathbb{F}_{p^{12}}$, say $\mathbb{F}_{p^{24}}$ for example, must now take into account the isomorphism between the two quadratic fields. Condition CE6 ensures that this implicit isomorphism is made explicit by adding a quadratic field $\mathbb{F}_{p^2}$ in the lattice and explicitly embedding it in $\mathbb{F}_{p^4}$ and $\mathbb{F}_{p^6}$, and by transitivity in $\mathbb{F}_{p^{12}}$ and $\mathbb{F}_{p^{24}}$.

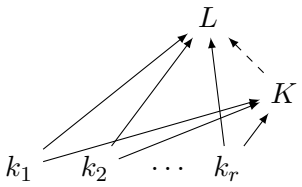
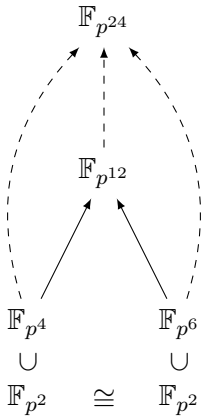


Figure 6.5: An incomplete diagram with several subfields.



Under the conditions described in Definition 6.3.4, we can add new embeddings in our lattice Λ without compromising the compatibility of the lattice. Assume that we want to embed K into L. We have seen that the relevant elements that restrict the number of compatible embeddings are the common subfields of K and L. If there is only one common subfield k, we are in the case described by Propositions 6.3.1 and 6.3.2. Now, assume that there are many common subfields, like shown in Figure 6.5. In the same fashion as was done in Proposition 6.3.1, we can abstractly describe how many compatible embeddings exist (this is done in [BCS97, Section 2.5]), but we rather focus on the constructive part. In some sense, this is just a generalization of Proposition 6.3.2, where we allow an arbitrary number of common subfields.

Proposition 6.3.5. *Let $K, L \in \mathcal{K}$ be two finite fields with*

$$K \subset L$$

and let $k_1, \ldots, k_r \in \mathcal{K}$ be the common subfields of K and L, i.e. *for all $1 \leq j \leq r$, we have*

$$k_j \hookrightarrow K \text{ and } k_j \hookrightarrow L.$$

Let $\alpha \in K$ be an element generating K over $\mathbb{F}_p$ and

$$\pi = \gcd_{1 \leq j \leq r} (\mathrm{Minpoly}_{k_j}(\alpha)).$$

Let $\rho \in L$ be a root of π in L and let $\psi : K \to L$ be the map defined by

$$\psi\left(\sum_{j=0}^{\deg(K)-1} x_j \alpha^j\right) = \sum_{j=0}^{\deg(K)-1} x_j \rho^j$$

using the unique representation of elements in K as $\sum_j x_j \alpha^j$ with $x_j \in \mathbb{F}_p$. Then ψ is an embedding from K to L that is compatible with all the embeddings $\phi_{k_j \hookrightarrow K}$ and $\phi_{k_j \hookrightarrow L}$.

Proof. Let $\alpha \in K$ be an element generating K over the base field $\mathbb{F}_p$. Then, for all $1 \leq j \leq r$, α also generates K over k_j. Now consider the polynomial

$$\pi = \gcd_{1 \leq j \leq r} (\mathrm{Minpoly}_{k_j}(\alpha)).$$

For all $1 \leq j \leq r$, we have by definition

$$(\mathrm{Minpoly}_{k_j}(\alpha))(\alpha) = 0$$

and so

$$T - \alpha \,|\, \mathrm{Minpoly}_{k_j},$$

thus the polynomial $T - \alpha \in K[T]$ also divides π, which consequently is not the constant polynomial 1 and has a root in K. Since we have

$$K \subset L,$$

it follows that π also has a root in L. Let $\rho \in L$ be a root of π in L, then for all $1 \leq j \leq r$, ρ is a root of the polynomial $\mathrm{Minpoly}_{k_j}(\alpha)$ and thus the map ψ is an embedding that is compatible, by Proposition 6.3.2 and Remark 6.3.3, with the embeddings $\phi_{k_j \hookrightarrow K}$ and $\phi_{k_j \hookrightarrow L}$. □

Remark 6.3.6. There is another way of describing the polynomial π in Proposition 6.3.5. Let $K' \subset K$ be the subfield of K generated by the fields $k_j \subset K$, in fact we have

$$\pi = \mathrm{Minpoly}_{K'}(\alpha).$$

Indeed, we have

$$\begin{aligned}
\mathrm{Gal}(K/K') &= \left\{\sigma \in \mathrm{Gal}(K/\mathbb{F}_p) \,|\, \forall x \in K',\, \sigma(x) = x\right\} \\
&= \{\sigma \in \mathrm{Gal}(K/\mathbb{F}_p) \,|\, \forall 1 \leq j \leq r,\, \forall x \in k_j,\, \sigma(x) = x\} \\
&= \bigcap_{1 \leq j \leq r} \mathrm{Gal}(K/k_j),
\end{aligned}$$

and thus it follows that

$$\begin{aligned}
\mathrm{Minpoly}_{K'}(\alpha) &= \prod_{\sigma \in \mathrm{Gal}(K/K')} T - \sigma(\alpha) \\
&= \prod_{\sigma \in \bigcap_{1 \leq j \leq r} \mathrm{Gal}(K/k_j)} T - \sigma(\alpha) \\
&= \gcd_{1 \leq j \leq r} \left(\prod_{\sigma \in \mathrm{Gal}(K/k_j)} T - \sigma(\alpha) \right) \\
&= \gcd_{1 \leq j \leq r} (\mathrm{Minpoly}_{k_j}(\alpha)).
\end{aligned}$$

where T is the indeterminate. In fact, in [BCS97], it is shown that there is exactly one compatible ismorphism between K' and L', the subfield of L generated by the subfields k_j in L, and that there are

$$\deg(K)/\deg(K'),$$

which is the degree of π, different compatible embeddings from K to L.

6.3.2 Implementation in Nemo

Although Proposition 6.3.1 suggests to compute a random embedding and then to "correct" it, we follow the method of Propositions 6.3.2 and 6.3.5 and we directly compute compatible embeddings using the naive algorithm, that relies on polynomial factorization. The whole framework is implemented in Nemo [H+16] since Fall 2019, at least for finite fields with word-sized characteristic, *i.e.* finite fields extensions

$$\mathbb{F}_{p^m}$$

where p is a prime number that fits in 64 bits. The source code is available in Nemo's github repository[1]. Almost all of the code is directly written in the Julia library Nemo, because it consists mostly of high level manipulations, but the critical routines, such as factorization, are implemented in the C library Flint and are called from Nemo.

Types and data structures. There are two types of finite fields in Nemo/Flint, those with a word-sized characteristic and those with arbitrarily large characteristic. The former have the type `fq_nmod_ctx_t` in Flint and `FqNmodFiniteField` in Nemo, while the field elements have the type `fq_nmod_t` in Flint and `fq_nmod` in Nemo. The corresponding types in arbitrarily large characteristic are `fq_ctx_t`, `FqFiniteField`, `fq_t` and `fq`. We may use a Flint name for a type in Nemo or vice versa, because the types represent the same things: Nemo being a wrapper of Flint in this case. The main difference between the two types is that the polynomials representing the elements of these fields have different types, depending on whether they have arbitrarily large coefficients (`fmpz_mod_poly_t`) or if their coefficients fit in a 64 bit word (`nmod_poly_t`). Since the representation of the elements varies, so do the algorithms for manipulating various elements in Flint (matrices, polynomials) with word-sized or arbitrarily large coefficients. Some of the algorithms implemented for matrices with word-sized coefficients are not implemented for arbitrarily large coefficient and as a consequence we were only able to implement the Bosma-Canon-Steel algorithm for the type `FqNmodFiniteField`. This type contains

- the characteristic p (that fits in 64 bits);
- the irreducible polynomial used to define the field (of type `nmod_poly_t`);
- various technical precomputations that are stored for performance;
- two dictionaries containing information about the lattice of compatible embeddings.

Let K be some finite field represented by the type `FqNmodFiniteField`, the two dictionaries stored in the data structure representing K are called `subfields` and `overfields`. They both map integers to lists of embeddings. If the `subfields` dictionary has a key `l`, then the associated value `subfields[l]` is a list of embeddings

$$\phi_j : k_j \to K$$

where all the k_j are finite fields of the corresponding degree `l`. Note that this is a *list*, because there might be several different finite fields of the same degree embedded in K. Similarly, if the `overfields` dictionary has a key `m`, then the corresponding value `overfield[m]` is a list of embeddings

$$\psi_j : K \to L_j$$

[1] https://github.com/Nemocas/Nemo.jl

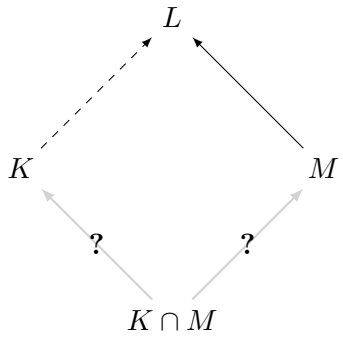
where all the L_j are finite fields of degree `m`. Finally, the embeddings $\phi : K \to L$ are represented by the type `FinFieldMorphism`, that essentially contains information about the domain K, the codomain L, the function ϕ and the preimage function ϕ^{-1}.

General strategy. The essential idea is to maintain a lattice of compatibly embedded finite fields, *i.e.* ensure that the conditions described in Definition 6.3.4 are met, at all times. To do so, each time the user asks for a new embedding

$$\phi : K \to L,$$

we must go through 3 steps:

1. for each subfield
$$M \subset L$$
of L, check that the finite field $M \cap K$ is embedded in M and K, and if not, embed it. If there is not any finite field of degree
$$d = \gcd(\deg(M), \deg(K)),$$
compute an arbitrary finite field I of degree d using Flint and embed I in M and K.



This step ensures that Condition CE6 on the *intersections* holds. It is important to begin with this step, so that every implicit isomorphism between the different common subfields are made explicit and that the compatibility conditions concerning the embedding $K \hookrightarrow L$ include them.

2. Embed K in L using the procedure described in Proposition 6.3.5.

3. Compute the "transitive closure" of the lattice, *i.e.* compute the embeddings such that Condition CE5 on *transitivity* holds.

The first step might contain a recursive call to the embedding algorithm, thus one might have to compute many additional embeddings behind the scenes in order to have only one new embedding.

Main algorithms. The embedding algorithm, called `embed`, thus follows the steps described in the general strategy, alongside trivial verifications such as checking if the embedding makes sense, checking if an embedding already exists, and so on. There are three main algorithms, `intersections`, `find_morphism` and `transitive_closure`, each one respectively corresponding to the first, the second, and the third step. In the first step, we loop through all the subfields M

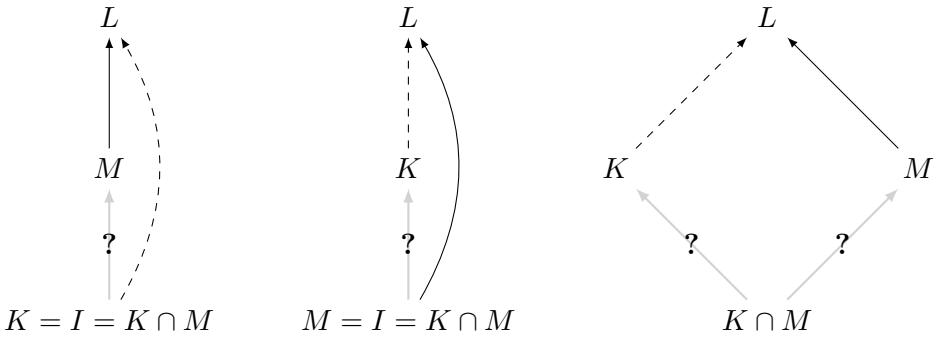


Figure 6.6: Three different configurations when computing the intersection step of the Bosma-Canon-Steel framework.

embedded in L and we check that the intersection $I = K \cap M$ is also embedded in K and M. Different cases can occur, depending on the relative position of K and M in the lattice of finite fields, as described in Figure 6.6. If $I = K$ (case of the left side of Figure 6.6), then we only need to check that K is embedded in M. When this is done, we do not need to compute anything else because the embedding from K to L is then obtained via transitive closure. If $I = M$ (case in the middle), then we need to check that M is embedded in K. If I is neither K or L (case on the right side), then we first check if I already exists in the lattice of compatibly embedded finite fields, we create it if necessary, and we finally check that it is embedded in both K and L. In order to know whether further computations are needed in the `embed` algorithm, we return a boolean that is `false` if the encountered case was the one where $I = K$. The `intersection` algorithm is also explained in Algorithm 8. In Algorithm 9, called `find_morphism`, we follow the method of Proposition 6.3.5 in order to find a compatible embedding. If there are no particular conditions to meet, we just use polynomial factorization to obtain an embedding, *i.e.* we use the naive embedding algorithm. When a suitable embedding $\phi : K \hookrightarrow L$ has been found, in order to ensure that the lattice Λ is transitive, we apply Algorithm 10 that loops through all subfields k of K, and letting $\psi : k \hookrightarrow K$ be an embedding into K, checks that the embedding $\phi \circ \psi$ is also in the lattice. We then recursively call the `transitive_closure` procedure on the fields that L is embedded into.

Experimental results. All the tests in this section were performed on an Intel Core i7-7500U CPU clocked at 2.70GHz, using Nemo 0.19.1 running on Julia 1.5.3, and Nemo's corresponding version of Flint. The benchmark functions are available in the file `bencharks.jl` of the repository of the thesis[2], as well as the data that was used to plot the timings. All plots were made using gnuplot version 5.2 patchlevel 8, and the gnuplot files can also be found in the repository. Because of how the Bosma-Canon-Steel framework works, in particular because of the condition on the intersections, the time needed to compute a specific embedding

$$\mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$$

heavily depends on the other embeddings in the lattice. Indeed, if there are no embeddings in the lattice, the embedding computation is in that case essentially the same as the factorization of a degree m polynomial in $\mathbb{F}_{p^n}[X]$. On the contrary, if a subfield $\mathbb{F}_{p^d}$ of degree $d \mid m$ exists in the lattice (see Figure 6.7 for an illustration) and is embedded in both $\mathbb{F}_{p^m}$ and $\mathbb{F}_{p^n}$, then the degree

[2] https://github.com/erou/thesis/.

Algorithm 8 `intersections`

Input: K and L two finite fields of a lattice of compatibly embedded finite fields $(\mathcal{K}, \Phi)$
Output: A boolean b being **false** if and only if no additional work is required

$b \leftarrow$ **true**
for all $M \hookrightarrow L$ **do**
  $d \leftarrow \gcd([K : \mathbb{F}_p], [M : \mathbb{F}_p])$
  if $d = [K : \mathbb{F}_p]$ **then**
    $b \leftarrow$ **false** ▷ We obtain the final embedding by transitive closure
    `embed`(K, M)
  else if $d = [M : \mathbb{F}_p]$ **then**
    `embed`(M, K)
  else if $I \cong \mathbb{F}_{p^d} \hookrightarrow K$ **then** ▷ $\mathbb{F}_{p^d}$ already exists in the computer algebra system
    `embed`(I, M)
  else if $I \cong \mathbb{F}_{p^d} \hookrightarrow L$ **then** ▷ $\mathbb{F}_{p^d}$ already exists in the computer algebra system
    `embed`(I, K)
    `embed`(I, M)
  else
    $I \leftarrow \mathbb{F}_{p^d}$ ▷ We create the field $\mathbb{F}_{p^d}$ in the computer algebra system
    `embed`(I, K)
    `embed`(I, M)
  end if
end for
return b

Algorithm 9 `find_morphism`

Input: K and L two finite fields of a lattice of compatibly embedded finite fields $(\mathcal{K}, \Phi)$
Output: $\phi : K \to L$ a compatible embedding

$C \leftarrow \{k \in \mathcal{K} \mid k \hookrightarrow K \text{ and } k \hookrightarrow L\}$
$\pi \leftarrow \gcd_{k \in C}(\text{Minpoly}_k(\alpha))$ ▷ α is a generator of K over $\mathbb{F}_p$, the minimum polynomial is obtained via the Berlekamp-Massey algorithm.
$\rho \leftarrow$ any root of π
return $\phi : K \to L$, $\alpha \mapsto \rho$

Algorithm 10 `transitive_closure`

Input: $\phi : K \hookrightarrow L$ an embedding between two finite fields
Output: ensures that the lattice of compatibly embedded finite fields $(\mathcal{K}, \Phi)$ is transitive

for all $\psi : k \hookrightarrow K$ **do**
  $\Phi \leftarrow \Phi \cup \{\phi \circ \psi\}$
end for
$C \leftarrow \{M \in \mathcal{K} \mid L \hookrightarrow M\}$
for all $M \in C$ **do**
  $\theta \leftarrow (L \hookrightarrow M)$
  `transitive_closure`(θ)
end for

of the polynomial that is factorized is at most $\frac{m}{d}$, and could be less if several subfields exist, as explained in Proposition 6.3.5 and Remark 6.3.6. As an illustration of that phenomenon, note

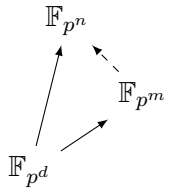


Figure 6.7: The computation of an embedding with a common subfield, that leads to the factorization of a polynomial of degree at most m/d.

that if the finite fields $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^3}$, $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$ are in our lattice, and if the embeddings

$$\mathbb{F}_{p^2} \hookrightarrow \mathbb{F}_{p^{12}}$$

and

$$\mathbb{F}_{p^3} \hookrightarrow \mathbb{F}_{p^{12}}$$

are already computed, then there is only one compatible embedding from $\mathbb{F}_{p^6}$ to $\mathbb{F}_{p^{12}}$. In this

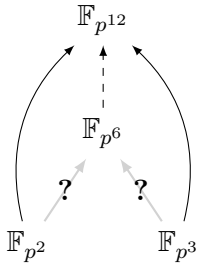


Figure 6.8: A lattice with four finite fields of size p^2, p^3, p^6, p^{12} with some embeddings already computed.

case, factorization in $\mathbb{F}_{p^{12}}[X]$ is not required, and the work is essentially already done, although the embeddings from $\mathbb{F}_{p^2}$ and $\mathbb{F}_{p^3}$ in $\mathbb{F}_{p^6}$ (easier to compute) have to be computed if they are not already (see Figure 6.8). In fact, when computing the embedding

$$\mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n},$$

one of the decisive parameters is the degree of the extension

$$\mathbb{F}_{p^m}/(L \cap \mathbb{F}_{p^m}),$$

where L is the finite field generated by the subfields already embedded in $\mathbb{F}_{p^n}$. This measures both the freedom that we have to choose the embedding $\mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$ and how much we already know about it. If

$$[\mathbb{F}_{p^m} : (L \cap \mathbb{F}_{p^m})] = 1,$$

then $\mathbb{F}_{p^m}$ is essentially already embedded in $\mathbb{F}_{p^n}$ and there is only one compatible embedding. On the contrary, if

$$[\mathbb{F}_{p^m} : (L \cap \mathbb{F}_{p^m})] = m,$$

i.e. if $L \cap \mathbb{F}_{p^m} = \mathbb{F}_p$, then there is absolutely no compatibility condition to meet and we can take any embedding we want, but at the same time we do not have any information about the embedding $\mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$. We call this parameter the *defect* of the embedding $\mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$. The defect depends on the state of the lattice Λ at the moment of the computation and has an important impact on the timings. Consequently, there are several reasonable ways of measuring the timings of the Bosma-Canon-Steel framework, which produce quite different results. There is no major difference between two choices of characteristic p so we arbitrarily chose $p = 3$. We computed all the possible embeddings between finite fields of degree up to 400 in two different ways. The first one consists in looping through the degrees while increasing them, *i.e.* for all $1 \leq m \leq 200$ and $m + 1 \leq n \leq 400$, we compute the embedding

$$\mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$$

whenever that makes sense, starting with $m = 1, n = 2$, then $m = 1, n = 3$, and so on, so that the last embedding we compute is

$$\mathbb{F}_{p^{200}} \hookrightarrow \mathbb{F}_{p^{400}}.$$

The second way consists in looping through the degrees while decreasing them, *i.e.* starting with $\mathbb{F}_{p^{200}} \hookrightarrow \mathbb{F}_{p^{400}}$, then $\mathbb{F}_{p^{199}} \hookrightarrow \mathbb{F}_{p^{398}}$ and so on until $\mathbb{F}_p \hookrightarrow \mathbb{F}_{p^2}$. Again, these two methods produce different timing results because at the moment an embedding is computed, the state of the lattice if not the same in the two cases. For example, the embedding $\mathbb{F}_{p^{200}} \hookrightarrow \mathbb{F}_{p^{400}}$ takes 3.5 seconds to be computed in the first test, while it takes 47.5 seconds in the second one. In Figure 6.9, we plot the time needed to compute embeddings of the form

$$\mathbb{F}_{p^2} \hookrightarrow \mathbb{F}_{p^m}$$

for $m \leq 400$ with the first method, *i.e.* increasing degrees. In this case, there are no common subfields when the embeddings are computed, thus the defect is always 1 and the naive embedding algorithm is applied. When the embeddings are computed with decreasing degrees, there are a lot more embeddings already in the lattice, thus the defect can be either 1 or 2. We observe that the time needed to compute the embeddings is completely different in those two cases, thus we use a logarithmic scale to emphasize the difference. The difference between the two methods is also noticeable when looking at the embeddings to a fixed finite field. We look at the embeddings to the finite fields of degree 360 because it is a highly composite number, having 24 different divisors: $1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 18, 20, 24, 30, 36, 40, 45, 60, 72, 90, 120, 180$ and 360. In the first experiment, with increasing degrees, all embeddings are computed in under one second, as shown in Figure 6.11. This is understandable because the defect is never very high. In the second experiment, shown in Figure 6.12, the timing to compute the embedding

$$\mathbb{F}_{p^{180}} \hookrightarrow \mathbb{F}_{p^{360}}$$

is approximately 30 seconds, with a defect of 180 because $\mathbb{F}_{p^{360}}$ has no embedded subfields yet. After this computation, only the embedding $\mathbb{F}_{p^{120}} \hookrightarrow \mathbb{F}_{p^{360}}$ has a defect equal to 2 and all the others have a defect equal to 1. This leads to very fast computation for some embeddings, that are essentially already computed. When computing embeddings corresponding to extension of a fixed degree, *e.g.* extensions of type

$$\mathbb{F}_{p^{2m}}/\mathbb{F}_{p^m},$$

the difference is also noteworthy: when the embeddings are computed with the degrees decreasing (Figure 6.14), then the defect is always maximum, thus the timings are rather smooth, while

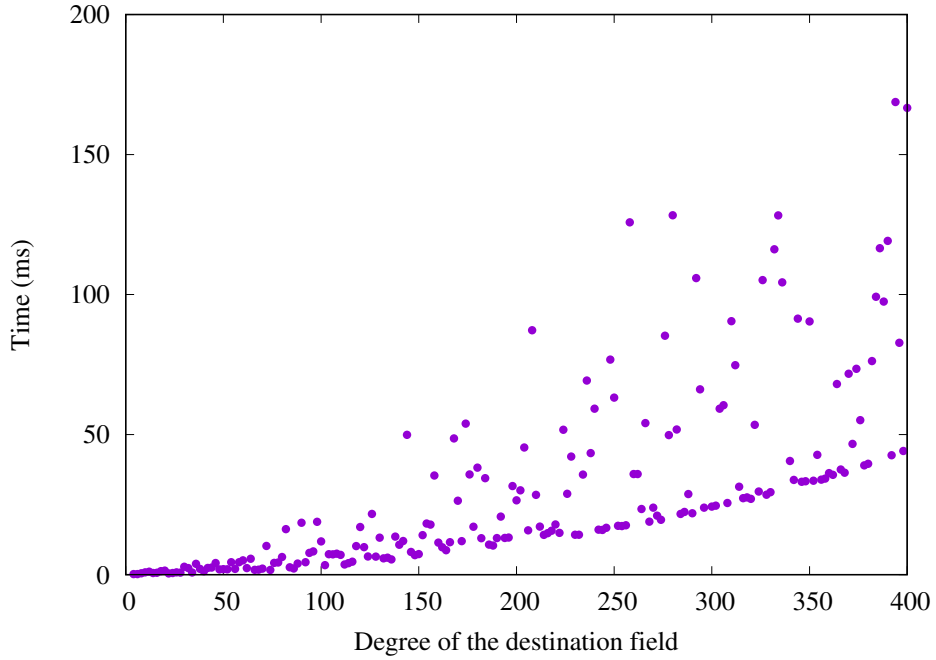


Figure 6.9: Timings for the computation of embeddings from $\mathbb{F}_{p^2}$ to $\mathbb{F}_{p^m}$ for $m \leq 400$ and $2 \mid m$, with $p = 3$. The embeddings were computed with increasing degrees.

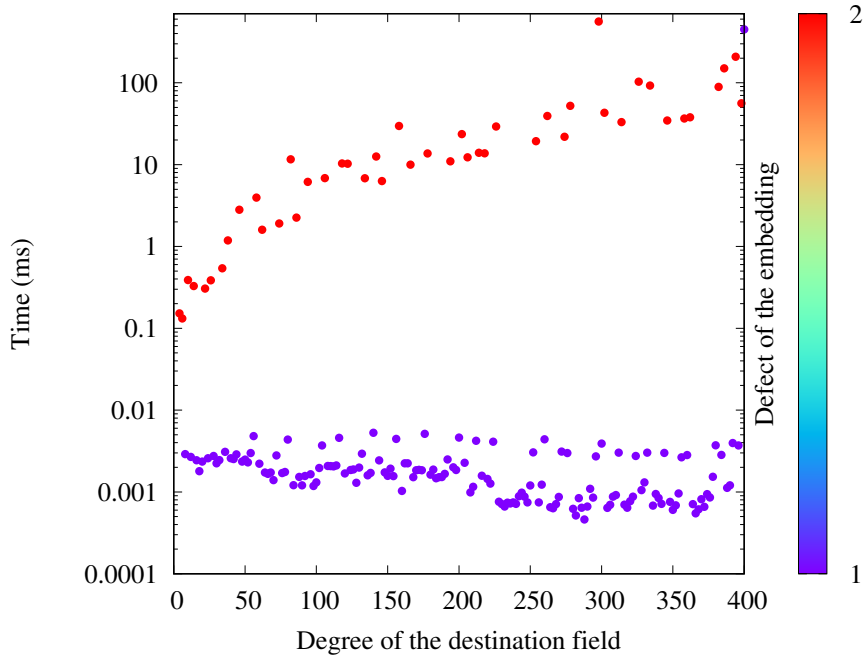


Figure 6.10: Timings (logarithmic scale) for the computation of embeddings from $\mathbb{F}_{p^2}$ to $\mathbb{F}_{p^m}$ for $m \leq 400$ and $2 \mid m$, with $p = 3$. The embeddings were computed with decreasing degrees.

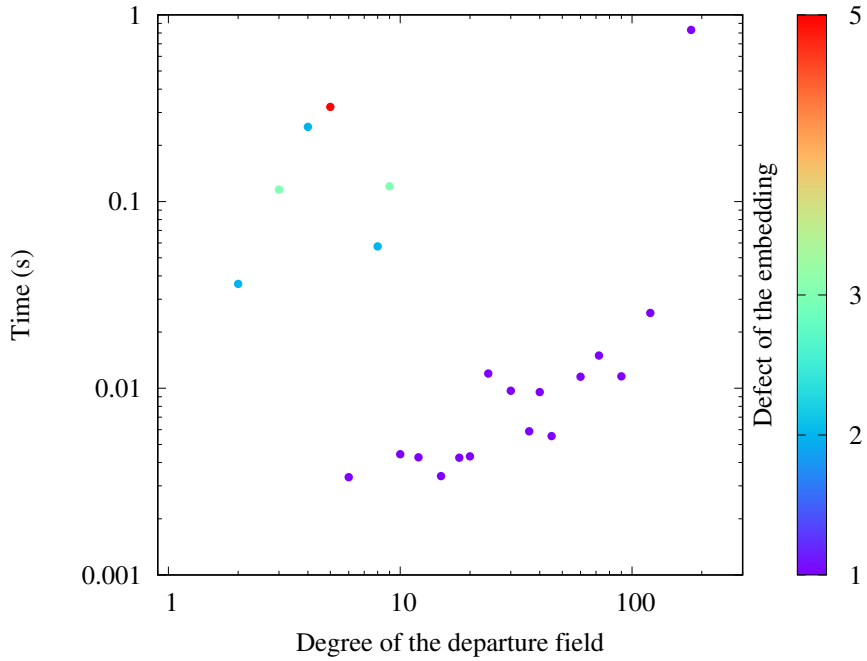


Figure 6.11: Timings for the computation of embeddings from $\mathbb{F}_{p^m}$ to $\mathbb{F}_{p^{360}}$ for $m \mid 360$, with $p = 3$. The embeddings were computed with increasing degrees. We use logarithmic scales on both axes for readability.

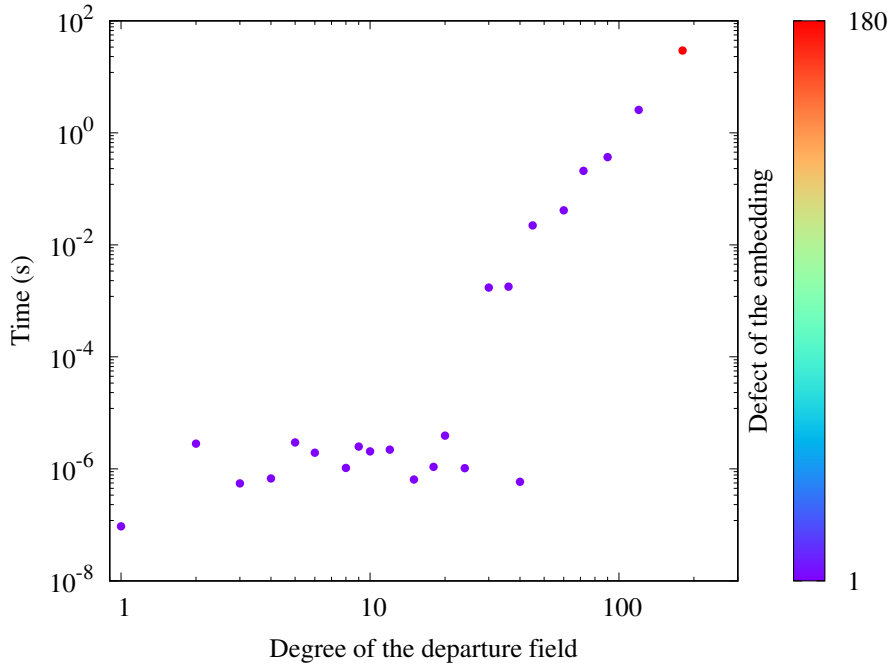


Figure 6.12: Timings for the computation of embeddings from $\mathbb{F}_{p^m}$ to $\mathbb{F}_{p^{360}}$ for $m \mid 360$, with $p = 3$. The embeddings were computed with decreasing degrees. We use logarithmic scales on both axes for readability.

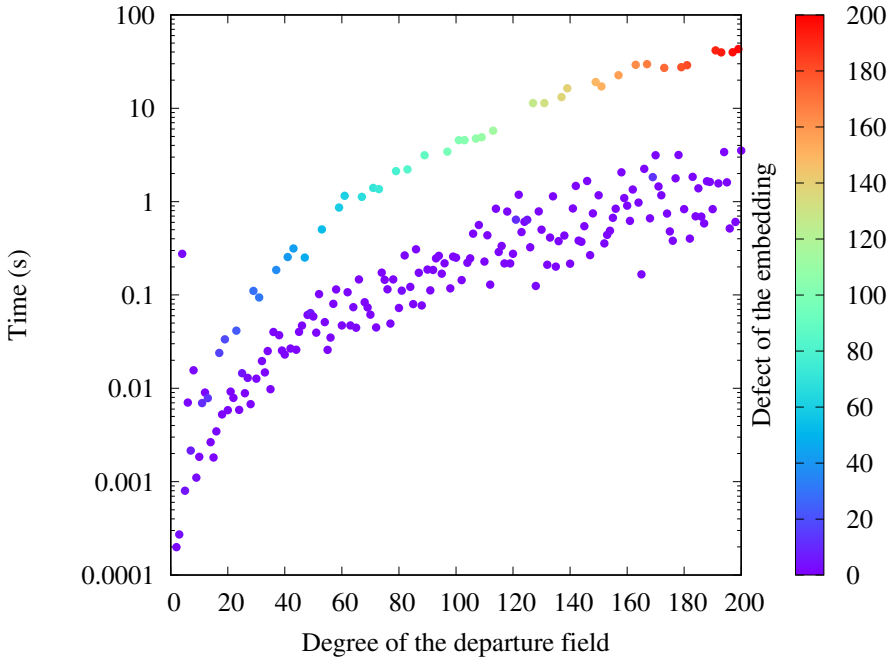


Figure 6.13: Timings (logarithmic scale) for the computation of embeddings from $\mathbb{F}_{p^m}$ to $\mathbb{F}_{p^{2m}}$ for $1 \leq m \leq 200$, with $p = 3$. The embeddings were computed with increasing degrees.

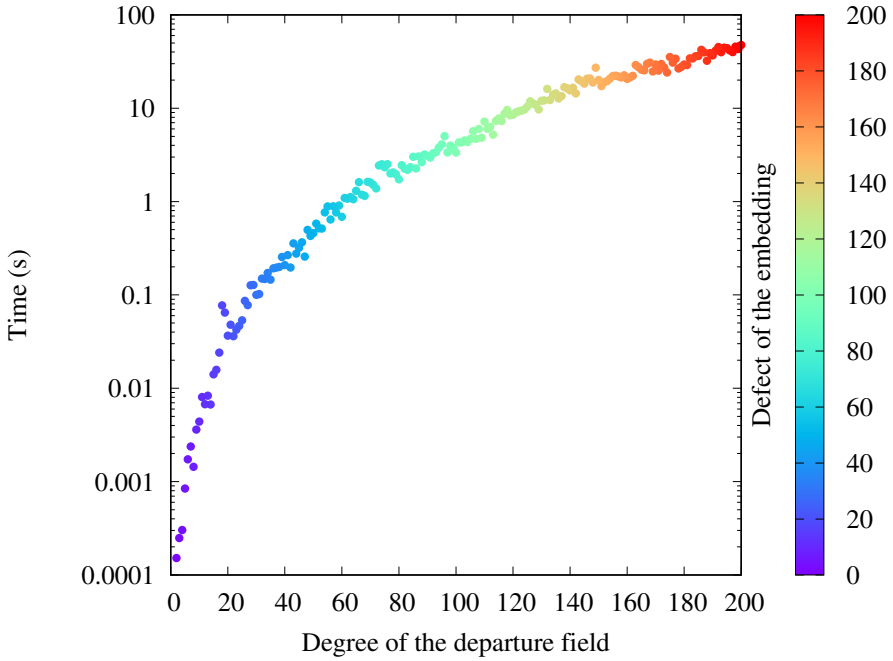


Figure 6.14: Timings (logarithmic scale) for the computation of embeddings from $\mathbb{F}_{p^m}$ to $\mathbb{F}_{p^{2m}}$ for $1 \leq m \leq 200$, with $p = 3$. The embeddings were computed with decreasing degrees.

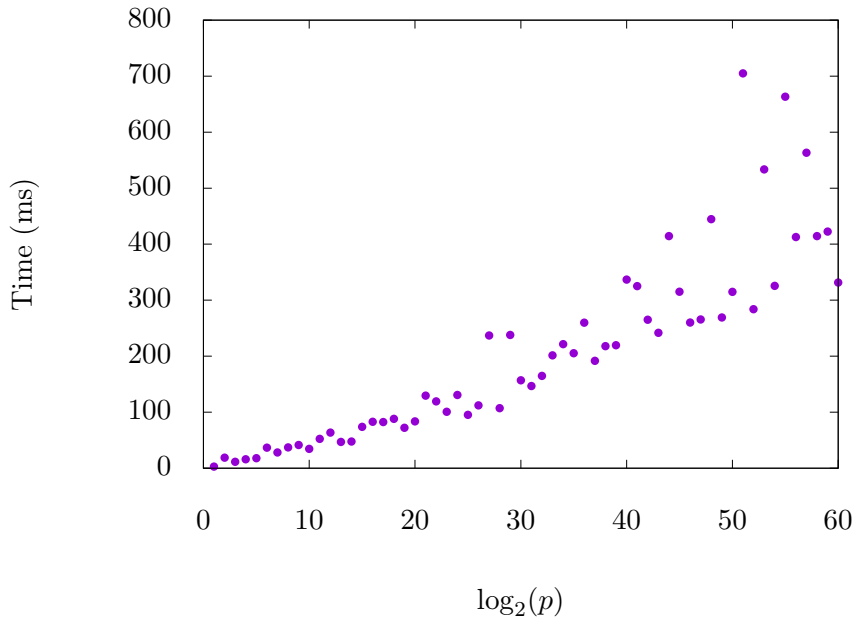


Figure 6.15: Timings for the computation of embeddings from $\mathbb{F}_{p^{12}}$ to $\mathbb{F}_{p^{24}}$ with p a prime number growing up to approximately 2^{60}.

the computations with the increasing degrees (Figure 6.13) produce erratic timings because low defects lead to a huge speedup. In order to show the impact on the characteristic p on the timings, we also plot the time needed to compute the embedding

$$\mathbb{F}_{p^{12}} \hookrightarrow \mathbb{F}_{p^{24}}$$

for different primes p. Because the impact on the timing is logarithmic in the characteristic p, we use each prime number found immediately after 2^j, for $1 \leq j \leq m$, *i.e.* $p \approx 2^j$. The result is shown in Figure 6.15. Whether or not compatibility conditions have to be fufilled when computing an embedding implies different internal routines in the Bosma-Canon-Steel framework, such as intersections computations and recursive computations of other embeddings. However, the different possible scenarios share the preponderance of the root finding in the timings: it is the critical routine in all cases. Although, for embeddings involving small degree finite fields and with a low defect (*i.e.* when other embeddings have already been computed and information is known) the time spent dealing with high level routines can be substantial. It is not really a problem because in that case the computations are very fast, but it indicates that the Julia code could be optimized some more.

Chapter 7

Standard lattice of compatibly embedded finite field

We have seen in Chapter 6 two independant methods to create lattices of compatibly embedded finite fields. In this chapter, we present a new framework, inspired by both Conway polynomials and the Bosma-Canon-Steel framework, that we call *standard lattice of compatibly embedded finite fields*.

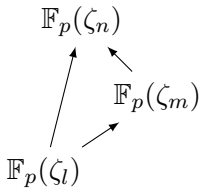
Contents

7.1 The Lenstra-Allombert algorithm and lattices of embeddings 132
7.1.1 From isomorphism to embedding 132
7.1.2 Cyclotomic lattices 134
7.1.3 Kummer embeddings 135
7.2 Standard solution of Hilbert 90 141
7.2.1 Complete algebras and standardization 141
7.2.2 Towards standard embeddings 146
7.3 Standard embeddings 148
7.4 Implementation 152
7.4.1 Complexity analysis 153
7.4.2 Experimental results 156

Outline of Chapter 7. In this chapter, we construct new theoretic tools to build a new framework to manage finite field extensions. Before studying these objects in details in the following sections, we briefly explain our general strategy. If we have three integers

$$l \mid m \mid n,$$

we can define embeddings between the finite fields $\mathbb{F}_p(\zeta_l), \mathbb{F}_p(\zeta_m)$ and $\mathbb{F}_p(\zeta_n)$, where ζ_l (resp. ζ_m, ζ_n) is a primitive l-th (resp. m-th, n-th) root of unity.

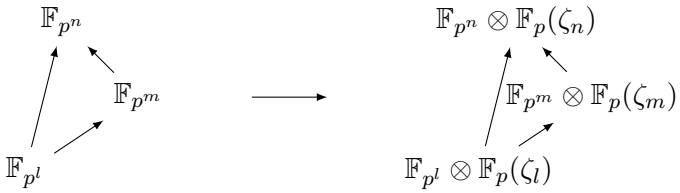


Indeed, one can send ζ_l to $(\zeta_m)^{m/l}$ or to $(\zeta_n)^{n/l}$ to embed $\mathbb{F}_p(\zeta_l)$ in either $\mathbb{F}_p(\zeta_m)$ or $\mathbb{F}_p(\zeta_n)$. Similarly, one can send ζ_m to $(\zeta_n)^{n/m}$ to obtain an embedding from $\mathbb{F}_p(\zeta_m)$ to $\mathbb{F}_p(\zeta_n)$. In order to achieve compatibility between these embeddings, one must have

$$((\zeta_n)^{n/m})^{m/l} = (\zeta_m)^{m/l} = \zeta_l.$$

This is similar to how Conway polynomials work: one ensures that every finite field can be described as $\mathbb{F}_p(\zeta)$ for some primitive root ζ that is compatible with every other primitive root. A simple way of obtaining such a compatible configuration is to start from the root ζ_n and compute ζ_l and ζ_m from ζ_n.

We see in Section 7.1 that this situation can be generalized to arbitrary extensions $\mathbb{F}_{p^l}, \mathbb{F}_{p^m}, \mathbb{F}_{p^n}$ using the Lenstra-Allombert algorithm. Starting from a primitive n-th root ζ_n, one can compute the roots ζ_l, ζ_m and construct the algebra $A_n = \mathbb{F}_{p^n} \otimes \mathbb{F}_p(\zeta_n)$, and the corresponding algebras A_l and A_m. Then, from a solution $\alpha_n \in A_n$ of (H90) for ζ_n, we can compute solutions α_l, α_m of (H90) in A_l and A_m, and deduce compatible embeddings between the finite fields $\mathbb{F}_{p^l}, \mathbb{F}_{p^m}, \mathbb{F}_{p^n}$.



Because we can deduce compatible embeddings from a solution α of (H90) in a large algebra, we study the largest algebra for a given field of scalars $\mathbb{F}_{p^a} = \mathbb{F}_p(\zeta_{p^a-1})$, *i.e.* the largest algebra for a given level a. We call this largest algebra the *complete algebra* of level a, and it is given by

$$A_{p^a-1} = \mathbb{F}_{p^{p^a-1}} \otimes \mathbb{F}_p(\zeta_{p^a-1}).$$

We see in Section 7.2.1 that the solutions α of (H90) for ζ_{p^a-1} in complete algebras all share a very special property: they satisfy

$$\alpha^{p^a-1} = (\zeta_{p^a-1})^a,$$

i.e. they all have the same Kummer constant. We also see that the solutions β of (H90) in smaller Kummer algebras of the same level a that are computed from α also share the same Kummer constant. Conversely, we can prove that the solutions that have the desired Kummer constant

$$c = (\zeta_{p^a-1})^a$$

are linked to a solution α in the complete algebra of level a. We thus call these special solutions *standard*, and we show in Section 7.2.2 how to construct compatible embeddings from them, that we call *standard embeddings*. Given a Kummer algebra $A_n = \mathbb{F}_{p^n} \otimes \mathbb{F}_p(\zeta_n)$ of degree a, we do not have to compute the complete algebra of degree a in order to find a standard solution in A_n, we can directly compute it in the smaller algebra A_n. We call the pair (A_n, α_n), where α_n is a *standard* solution, a *decorated algebra*. Decorated algebras allow us to compute embeddings in an *incremental* way, which is important when managing finite field extensions. We finally show how to link complete algebras of different levels in Section 7.3, so that two standard solutions of (H90) in two arbitrary Kummer algebras can always be used to compute a compatible embedding between the associated finite fields. Again, in the general case of two arbitrary Kummer algebras, the computed embedding is called *standard*, and it is also compatible with future embeddings, meaning that the framework is incremental. We then discuss the implementation of the framework in Section 7.4.

7.1 The Lenstra-Allombert algorithm and lattices of embeddings

The two methods of Chapter 6 both have their drawbacks: Conway polynomials are expensive to compute and thus need to be precomputed, making them inefficient for large extensions, while the Bosma-Canon-Steel framework needs more computation each time an embedding is added to the lattice. Our starting point in order to propose an alternative framework for lattices of compatibly embedded finite fields is the Lenstra-Allombert algorithm and the study of Kummer algebras done in Section 5.2.2. In all this chapter, $\mathbf{k} = \mathbb{F}_p$ is a finite field of size p, where p is a prime number.

7.1.1 From isomorphism to embedding

Let us first recall the Lenstra-Allombert *isomorphism* algorithm. We keep the notations of Section 5.2, where the details can be found. Let K and L be two finite fields with p^n elements, where $\gcd(p, n) = 1$, *i.e.*

$$p \nmid n.$$

We know that K and L are isomorphic and, if ζ is a primitive n-th root of unity taken in the algebraic closure $\bar{\mathbb{F}}_p$ of $\mathbf{k}$, we know we can find an isomorphism by finding two solutions α_K and α_L to the equation (H90)

$$(\sigma \otimes 1)(\alpha) = (1 \otimes \zeta)\alpha,$$

respectively in $K \otimes \mathbb{F}_p(\zeta)$ and $L \otimes \mathbb{F}_p(\zeta)$. We then compute $\kappa \in \mathbb{F}_p(\zeta)$ such that

$$1 \otimes \kappa^n = \alpha_K^n / \alpha_L^n$$

and the map

$$\phi : \lfloor \alpha_K \rfloor_\zeta \mapsto \lfloor (1 \otimes \kappa)\alpha_L \rfloor_\zeta$$

is then an isomorphism from K to L. A key part of the algorithm is that the root ζ must be the same in the two Kummer algebras $K \otimes \mathbb{F}_p(\zeta)$ and $L \otimes \mathbb{F}_p(\zeta)$. In practice, it means that we need to use elements that have the same minimal polynomial to define ζ in both algebras. This constraint might seem easy to fulfill in this case, but it becomes harder in the case of a *compatible embedding* computation. Assume that $m, n \in \mathbb{N}$ are two integers such that

$$m \mid n$$

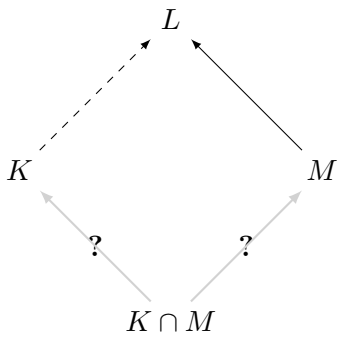
and $\gcd(p, m) = \gcd(p, n) = 1$. Let K be a finite field with p^m elements and L a finite field with p^n elements. We know that K is isomorphic to a subfield of L, *i.e.* we have an embedding

$$K \hookrightarrow L.$$

To compute an embedding, one solution is to compute the algebras $K \otimes \mathbb{F}_p(\zeta_m)$ and $L \otimes \mathbb{F}_p(\zeta_m)$, where ζ_m is a primitive m-th root of unity, as done in the isomorphism case, then compute solutions $\alpha_{K,m}, \alpha_{L,m}$ of (H90) and the constant $\kappa = \kappa_{K \hookrightarrow L}$. This solution is satisfying as long as we only want to compute a *single* embedding in L. Indeed, assume we also have an integer $l \in \mathbb{N}$ that divides n, such that $\gcd(p, l) = 1$, and a finite field H of size p^l. Then there is an embedding

$$H \hookrightarrow L,$$

and in order to compute it we must find a primitive l-th root of unity ζ_l, compute $L \otimes \mathbb{F}_p(\zeta_l)$, compute a new solution $\alpha_{L,l}$ of (H90) for ζ_l and the associated constant $\kappa_{H \hookrightarrow L}$. Therefore, each new embedding comes with the computation of a new Kummer algebra, a new solution of (H90), and a new element κ. We must also store the elements κ and the elements defining the embeddings. Now, recall that if we want to use the Bosma-Canon-Steel framework in order to compatibly embed K in L, we must recursively embed the intersection $K \cap M$ in both fields K and M, for each already embedded subfield M of L.



This yields a quadratic memory complexity in the number of extensions in the lattice and their degrees, as well as a quadratic number of new embedding computations, *i.e.* computations of Kummer algebras and solutions of (H90). It motivates a new solution with only one computation of Kummer algebra and (H90) solution per extension in the lattice, independently of the number of embedded subfields. Assume we have ζ_m and ζ_n respectively two m-th and n-th primitive roots of unity that are *compatible*, *i.e.* such that

$$(\zeta_n)^{n/m} = \zeta_m.$$

We compute the Kummer algebras $K \otimes \mathbb{F}_p(\zeta_m)$ and $L \otimes \mathbb{F}_p(\zeta_n)$, α_K a solution of (H90) for the root ζ_m and α_L a solution of (H90) for the root ζ_n. In that case, the element

$$(\alpha_L)^{n/m} \in L \otimes \mathbb{F}_p(\zeta_n)$$

is a solution of (H90) for the root $(\zeta_n)^{n/m} = \zeta_m$, indeed

$$\begin{aligned}(\sigma \otimes 1)((\alpha_L)^{n/m}) &= ((\sigma \otimes 1)(\alpha_L))^{n/m} \\ &= ((1 \otimes \zeta_n)\alpha_L)^{n/m} \\ &= (1 \otimes (\zeta_n)^{n/m})(\alpha_L)^{n/m}.\end{aligned}$$

The embedding $K \hookrightarrow L$ is then described by

$$\lfloor \alpha_K \rfloor_{\zeta_m} \mapsto \left\lfloor (1 \otimes \kappa_{K \hookrightarrow L})(\alpha_L)^{n/m} \right\rfloor_{(\zeta_n)^{n/m}},$$

where $\kappa_{K \hookrightarrow L} \in \mathbb{F}_p(\zeta_n)$ is a m-th root of α_L^n / α_K^m. There are still two issues with such a solution. First, it is still necessary to store the constants $\kappa_{K \hookrightarrow L}$ for each embedding

$$K \hookrightarrow L$$

in the lattice. We would like these constants κ to be equal to 1, or maybe that a close formula exists for these constants, by choosing special solutions α of (H90). We achieve the latter by constructing *standard* solutions of (H90) in Section 7.2.

7.1.2 Cyclotomic lattices

The second, and most important, issue is the compatibility condition between the roots of unity ζ. When we write a compatibility condition like

$$\zeta_m = (\zeta_n)^{n/m},$$

we implicitly state that there is a natural inclusion

$$\mathbb{F}_p(\zeta_m) \subseteq \mathbb{F}_p(\zeta_n)$$

that makes the embedding from $\mathbb{F}_p(\zeta_m)$ to $\mathbb{F}_p(\zeta_n)$ trivial, *i.e.* the embedding is the identity in that case. In practice, this is not always the situation at hand. For example, if for some reason the root ζ_m already exists in some field $\mathbb{F}_{p^a}$ in the current state of our computer algebra system, and if the root ζ_n lives in a strictly bigger field $\mathbb{F}_{p^b} = \mathbb{F}_p(\zeta_n)$ that we have to compute, then the field $\mathbb{F}_{p^a}$ is not included in the field $\mathbb{F}_{p^b}$, and the embedding

$$\mathbb{F}_{p^a} \hookrightarrow \mathbb{F}_{p^b}$$

is not trivial. In the general case, if we want to use the Lenstra-Allombert embedding algorithm, what we need is a *cyclotomic lattice*, given by Definition 7.1.1.

Definition 7.1.1 (Cyclotomic lattice)**.** A *cyclotomic lattice* is composed of two things:

- a collection
$$\mathcal{S}^I = \{(K_m, \zeta_m)\}_{m \in I}$$
over some support set $I \subset \mathbb{N} \setminus p\mathbb{N}$. The element K_m is an explicitly represented finite extension of $\mathbf{k} = \mathbb{F}_p$, and the element $\zeta_m \in K_m$ is a generating element of K_m that is also a primitive m-th root of unity, *i.e.* we have
$$K_m = \mathbb{F}_p(\zeta_m)$$
and
$$(\zeta_m)^m = 1.$$
- explicit embeddings
$$\begin{array}{rccc} \iota_{m,n} : & K_m & \hookrightarrow & K_n \\ & \zeta_m & \mapsto & (\zeta_n)^{n/m} \end{array}$$
whenever $(m, n) \in I^2$ are such that $m \mid n$.

Again, there is no problem if we know beforehand all the degrees of the extensions in the lattice that we will use, *i.e.* if the support set I is finite. Indeed, in that case there is an efficient randomised algorithm to compute the cyclotomic lattice: consider

$$N = \mathrm{lcm}_{m \in I}(m)$$

and construct the smallest finite field $\mathbb{F}_{p^a}$ such that N divides $p^a - 1$, *i.e.* the smallest finite field containing an N-th primitive root of unity. Then take $x \in \mathbb{F}_{p^a}$ at random, compute

$$y = x^{(p^a-1)/N}$$

and check that the multiplicative order of y is N. If it is, we can construct all roots ζ_m as powers of this element:

$$\zeta_m = y^{N/m}$$

for all $m \in I$, and we can set

$$K_m = \mathbb{F}_p(\zeta_m) \subset \mathbb{F}_{p^a}$$

and let the embeddings $\iota_{m,n}$ be natural inclusions. But once again, this methode does not produce an incremental lattice, thus it is not really user friendly: one would like to have a lattice where new elements can be added on the fly. Conway polynomials, that were introduced in Section 6.2 in order to construct a lattice of compatibly embedded finite fields, offer an other example of cyclotomic lattice. In fact, a cyclotomic lattice is always a lattice of compatibly embedded finite fields, because each time we have $l, m, n \in I$ with

$$l \mid n \mid n,$$

we have

$$(\zeta_n)^{n/l} = ((\zeta_n)^{m/l})^{n/m}$$

and it follows that

$$\iota_{l,n} = \iota_{m,n} \circ \iota_{l,m}.$$

One can thus wonder why we need a structure than can be used to represent a lattice of compatibly embedded finite fields, precisely to construct a lattice of compatibly embedded finite fields. In fact, we will see in the next sections that with a fairly small cyclotomic lattice, we are able to construct a much larger lattice of compatibly embedded finite fields, thus making the whole construction interesting, above all if the cyclotomic lattice is incremental, like with Conway polynomials. In the next sections, we consider that we have an abstract cyclotomic lattice, without specifying any particular construction. We only assume that we have a collection $\mathcal{S}^I$ satisfying the conditions of Definition 7.1.1.

7.1.3 Kummer embeddings

As we have seen in the last sections, asking for a compatibility condition

$$\zeta_m = (\zeta_n)^{n/m}$$

each time we want to use the Lenstra-Allombert embedding algorithm to embed $\mathbb{F}_{p^m}$ in $\mathbb{F}_{p^n}$, in a compatible way, is not trivial: it requires the availability of a cyclotomic lattice. Moreover, this equation implies that there is a natural inclusion

$$\mathbb{F}_p(\zeta_m) \subset \mathbb{F}_p(\zeta_n),$$

which is not the case in general. In order to be as thorough as possible, we will thus write the image of ζ_m in the larger field $\mathbb{F}_p(\zeta_n)$ as $\iota_{m,n}(\zeta_m)$. By definition, we have

$$\iota_{m,n}(\zeta_m) = (\zeta_n)^{n/m}.$$

We then generalize the discussion of Section 7.1.1 and the results of Section 5.2.3 in this setting. We keep the "Kummer algebra" terminology, already used in Section 5.2.2, that is based on [DFRR19]. We now assume that a cyclotomic lattice $\mathcal{S}^I$ is available. Let

$$m \mid n$$

be two integers prime to p, we then have an embedding

$$\begin{array}{rccc} \iota_{m,n}: & \mathbb{F}_p(\zeta_m) & \hookrightarrow & \mathbb{F}_p(\zeta_n) \\ & \zeta_m & \mapsto & (\zeta_n)^{n/m}. \end{array}$$

We also let

$$A_m = \mathbb{F}_{p^m} \otimes \mathbb{F}_p(\zeta_m)$$

and

$$A_n = \mathbb{F}_{p^n} \otimes \mathbb{F}_p(\zeta_n)$$

be two Kummer algebras. As was the case for the Lenstra-Allombert *isomorphism* algorithm, we want to deduce a field embedding from an algebra embedding between A_m and A_n, using the properties of the solutions of (H90). We are thus interested in a special class of morphisms that are closely linked with these solutions.

Definition 7.1.2 (Kummer embedding)**.** A *Kummer embedding* of A_m into A_n is an injective $\mathbf{k}$-algebra morphism

$$\Phi : A_m \hookrightarrow A_n$$

such that:

- the morphism Φ extends the scalar embedding $1 \otimes \iota_{m,n}$;
- the morphism Φ commutes with $\sigma \otimes 1$.

We can in fact give a simpler characterization of Kummer embeddings, and see that they are of the form $\Phi = \phi \otimes \iota$, where ι is the embedding described by the cyclotomic lattice $\mathcal{S}^I$. The embedding ϕ is then the one for which we will try to obtain a description, using the properties of the solutions of (H90).

Proposition 7.1.3. *There is a* 1*-to-*1 *correspondence between Kummer embeddings*

$$\Phi : A_m \hookrightarrow A_n$$

and embeddings of finite fields

$$\phi : \mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n},$$

given by

$$\Phi = \phi \otimes \iota_{m,n} \longleftrightarrow \phi.$$

Moreover, this correspondence commutes with composition of embeddings.

Proof. We will prove that the correspondence is given by:

- if Φ is a Kummer embedding, then Φ maps $\mathbb{F}_{p^m} \otimes 1$ into $\mathbb{F}_{p^n} \otimes 1$. Thus the restriction of Φ to $\mathbb{F}_{p^m}$ is of the form $\phi \otimes 1$ for some embedding $\phi : \mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$, and we have $\Phi = \phi \otimes \iota_{m,n}$;
- conversely, if $\phi : \mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$ is an embedding of finite fields, then $\Phi = \phi \otimes \iota_{m,n}$ is a Kummer embedding.

Let $\Phi : A_m \hookrightarrow A_n$ be a Kummer embedding. Since Φ is an algebra morphism, we have

$$\Phi(\beta^p) = \Phi(\beta)^p$$

for all $\beta \in A_m$. Since $(\sigma \otimes \sigma)(\beta) = \beta^p$, this proves that Φ commutes with $\sigma \otimes \sigma$. It also commutes with $\sigma \otimes 1$, and thus with its inverse $\sigma^{-1} \otimes 1$. It then also commutes with

$$(\sigma^{-1} \otimes 1) \circ (\sigma \otimes \sigma) = 1 \otimes \sigma.$$

Now, if $\beta \in \mathbb{F}_{p^m} \otimes 1$, we know thanks to Remark 5.2.7 that it is fixed by $1 \otimes \sigma$, thus we have that

$$\begin{aligned}(1 \otimes \sigma) \circ \Phi(\beta) &= \Phi \circ (1 \otimes \sigma)(\beta) \\ &= \Phi(\beta).\end{aligned}$$

Again, using Remark 5.2.7, we then know that $\Phi(\beta) \in \mathbb{F}_{p^n} \otimes 1$. This proves that $\mathbb{F}_{p^m} \otimes 1$ is mapped into $\mathbb{F}_{p^n} \otimes 1$. Now, this means that every element of the form $x \otimes 1$ with $x \in \mathbb{F}_{p^m}$ is mapped to an element of the form $\Phi(x \otimes 1) = y \otimes 1$ with $y \in \mathbb{F}_{p^m}$. Because Φ is a morphism of algebras, if we let $\phi(x) = y$, we can check that

$$\phi : \mathbb{F}_{p^m} \to \mathbb{F}_{p^n}$$

is also a morphism. Therefore, the restriction of Φ on $\mathbb{F}_{p^m}$ is of the form $\phi \otimes 1$, where $\phi : \mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$ is an embedding of finite fields. We conclude that $\Phi = \phi \otimes \iota_{m,n}$. Indeed, if

$$\beta = \sum_j x_j \otimes y_j$$

is an element of the Kummer algebra A_m, we then have

$$\begin{aligned}\Phi(\beta) &= \Phi(\sum_j x_j \otimes y_j) \\ &= \sum_j \Phi(x_j \otimes y_j) \\ &= \sum_j \Phi(x_j \otimes 1) \times \Phi(1 \otimes y_j) \\ &= \sum_j (\phi \otimes 1)(x_j \otimes 1) \times (1 \otimes \iota_{m,n})(1 \otimes y_j) \\ &= \sum_j (\phi(x_j) \otimes 1) \times (1 \otimes \iota_{m,n}(y_j)) \\ &= \sum_j \phi(x_j) \otimes \iota_{m,n}(y_j) \\ &= \sum_j (\phi \otimes \iota_{m,n})(x_j \otimes y_j) \\ &= (\phi \otimes \iota_{m,n})(\sum_j x_j \otimes y_j),\end{aligned}$$

and thus

$$\Phi(\beta) = (\phi \otimes \iota_{m,n})(\beta)$$

for every element $\beta \in A_m$.

Conversely, if $\phi : \mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$ is an embedding of finite fields and if we define

$$\Phi = \phi \otimes \iota_{m,n},$$

we see that Φ is a morphism of $\mathbf{k}$-algebras that extends the scalar embedding $\iota_{m,n}$ by definition. The embedding ϕ is a power of the Frobenius automorphism σ and thus commutes with σ, hence $\sigma \otimes 1$ commutes with $\Phi = \phi \otimes \iota_{m,n}$, and this proves that Φ is a Kummer embedding.

Now, if we have three Kummer algebras A_l, A_m, A_n such that

$$l \mid m \mid n$$

and two Kummer embeddings $\Phi_{l,m} : A_l \hookrightarrow A_m$ and $\Phi_{m,n} : A_m \hookrightarrow A_n$, we know that there exists $\phi_{l,m} : \mathbb{F}_{p^l} \hookrightarrow \mathbb{F}_{p^m}$ and $\phi_{m,n} : \mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$ such that we have the following diagram.

$$\begin{array}{c} A_n \\ \uparrow \quad \Phi_{m,n} = \phi_{m,n} \otimes \iota_{m,n} \\ A_m \\ \uparrow \quad \Phi_{l,m} = \phi_{l,m} \otimes \iota_{l,m} \\ A_l \end{array}$$

Now the map $\Phi_{m,m} \circ \Phi_{l,m}$ is a Kummer embedding from A_l into A_n, hence there exist an embedding $\phi : \mathbb{F}_{p^l} \hookrightarrow \mathbb{F}_{p^n}$ such that

$$\Phi_{m,m} \circ \Phi_{l,m} = \phi \otimes \iota_{l,n}.$$

But we also have

$$\Phi_{m,m} \circ \Phi_{l,m} = (\phi_{m,n} \circ \phi_{l,m}) \otimes (\iota_{m,n} \circ \iota_{l,m}),$$

and since $\iota_{l,n} = \iota_{m,n} \circ \iota_{l,m}$ by definition, we obtain

$$\phi_{m,n} \circ \phi_{l,m} = \phi,$$

thus the correspondence commutes with compositions of embeddings. □

Proposition 7.1.4. *Let $\alpha_m \in A_m$ be a nonzero solution of (H90) for ζ_m, and let c_m be its Kummer constant. Then, there is a 1-to-1 correspondence between Kummer embeddings*

$$\Phi : A_l \hookrightarrow A_m$$

and solutions $\hat{\alpha} \in A_n$ of (H90) for $(\zeta_n)^{n/m} = \iota_{m,n}(\zeta_m)$ that also satisfy

$$\hat{\alpha}^m = 1 \otimes \iota_{m,n}(c_m).$$

The correspondence is given by

$$\Phi(\alpha_m) = \hat{\alpha}.$$

Proof. Let $\Phi : A_m \hookrightarrow A_n$ be a Kummer embedding. Lemma 5.2.8 shows that α_m is a generator of A_m as an $1 \otimes \mathbb{F}_p(\zeta_m)$ algebra. Thus every element $\beta \in A_m$ can be written in the form

$$\beta = \sum_{j=0}^{m-1} (1 \otimes b_j)(\alpha_m)^j,$$

and we obtain

$$\Phi(\beta) = \sum_{j=0}^{m-1} (1 \otimes \iota_{m,n}(b_j))\Phi(\alpha_m)^j,$$

therefore we see that Φ is determined by its image $\Phi(\alpha_m)$. Moreover, $\hat{\alpha} = \Phi(\alpha_m)$ is a solution of (H90) for $(\zeta_n)^{n/m} = \iota_{m,n}(\zeta_m)$ that satisfies $\hat{\alpha}^m = \iota_{m,n}(c_m)$. Indeed, this is a generalization of the computations done in Section 5.2.2 and a consequence of Definition 7.1.2. We have

$$\begin{aligned}(\sigma \otimes 1)(\hat{\alpha}) &= (\sigma \otimes 1)(\Phi(\alpha_m)) \\ &= \Phi((\sigma \otimes 1)(\alpha_m)) \\ &= \Phi((1 \otimes \zeta_m)\alpha_m) \\ &= (1 \otimes \iota_{m,n})(\zeta_m)\Phi(\alpha_m) \\ &= (1 \otimes \iota_{m,n}(\zeta_m))\hat{\alpha}\end{aligned}$$

and

$$\begin{aligned}\hat{\alpha}^m &= \Phi(\alpha_m)^m \\ &= \Phi(\alpha_m^m) \\ &= \Phi(1 \otimes c_m) \\ &= 1 \otimes \iota_{m,n}(c_m).\end{aligned}$$

Converserly, if $\hat{\alpha}$ is a solution of (H90) for $\iota_{m,n}(\zeta_m)$ such that $\hat{\alpha}^m = \iota_{m,n}(c_m)$, we see that

$$\Phi(\beta) = \sum_{j=0}^{m-1} (1 \otimes \iota_{m,n}(b_j))\hat{\alpha}^j,$$

for any element

$$\beta = \sum_{j=0}^{m-1} (1 \otimes b_j)(\alpha_m)^j$$

gives a well-defined morphism of algebras from A_m into A_n that satisfies $\Phi(\alpha) = \hat{\alpha}$ and the conditions in Definition 7.1.2, *i.e.* it extends $\iota_{m,n}$ and commutes with $\sigma \otimes 1$. □

With these two correspondences, we can now describe a little more the link between solutions of (H90) and the finite field embeddings ϕ that we compute from them.

Corollary 7.1.5. *Let $\alpha_m \in A_m$ be a nonzero solution of (H90) for ζ_m with Kummer constant c_m and let $\hat{\alpha} \in A_n$ be a solution of (H90) for $(\zeta_n)^{n/m}$ that satisfies $\hat{\alpha}^m = \iota_{m,n}(c_m)$. Then*

- *the solution $\hat{\alpha}$ belongs to the subset $\mathbb{F}_{p^l} \otimes \mathbb{F}_p((\zeta_n)^{n/m}) \subset A_n$;*
- *the assignation $\lfloor \alpha_m \rfloor_{\zeta_m} \mapsto \lfloor \hat{\alpha} \rfloor_{(\zeta_n)^{n/m}}$ defines an embedding $\phi : \mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$;*

- *the map* $\Phi = \phi \otimes \iota_{m,n}$ *is the unique Kummer embedding such that* $\Phi(\alpha_m) = \hat{\alpha}$.

Proof. By Proposition 7.1.4, we know that there exists a unique Kummer embedding

$$\Phi : A_m \hookrightarrow A_n$$

such that $\Phi(\alpha_m) = \hat{\alpha}$. We also know thanks to Proposition 7.1.3 that

$$\Phi = \phi \otimes \iota_{m,n}$$

for some embedding of finite fields $\phi : \mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$. If $\alpha_m = \sum_{j=0}^{a-1} x_j \otimes (\zeta_m)^j$, where a is the level of A_m, we obtain

$$\hat{\alpha} = \sum_{j=0}^{a-1} \phi(x_j) \otimes (\zeta_n)^{\frac{in}{m}},$$

thus we have $\hat{\alpha} \in \mathbb{F}_{p^m} \otimes \mathbb{F}_p((\zeta_n)^{n/m})$. We also see that $x_0 = \lfloor \alpha_m \rfloor_{\zeta_m}$ is mapped to $\phi(x_0) = \lfloor \hat{\alpha} \rfloor_{(\zeta_n)^{n/m}}$, but $\lfloor \alpha_m \rfloor_{\zeta_m}$ is a generating element of $\mathbb{F}_{p^m}$ by Proposition 5.2.10, hence the assignation

$$\lfloor \alpha_m \rfloor_{\zeta_m} \mapsto \lfloor \hat{\alpha} \rfloor_{(\zeta_n)^{n/m}}$$

defines ϕ. □

With these results we are ready to generalize Proposition 5.2.13 to the embedding case, with the cyclotomic lattice setting, giving a minor variation of the original algorithm of Allombert.

Algorithm 11 (Allombert's algorithm)

Input: $\mathbb{F}_{p^m}, \mathbb{F}_{p^n}$, for $m \mid n$ integers prime to p, and a cyclotomic lattice $\mathcal{S}^{\{l,m\}}$.
Output: $s \in \mathbb{F}_{p^m}, t \in \mathbb{F}_{p^n}$, such that the assignation $s \mapsto t$ defines an embedding $\phi : \mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$.

1: Construct the Kummer algebras A_m and A_n.
2: Find $\alpha_m \in A_m$ and $\alpha_n \in A_n$, nonzero solutions of (H90) for ζ_m and ζ_n respectively.
3: Compute their Kummer constants: $(\alpha_m)^m = 1 \otimes c_m$ and $(\alpha_n)^n = 1 \otimes c_n$.
4: Compute κ, a m-th root of $\iota_{m,n}(c_m)/c_n$.
5: Return $\lfloor \alpha_m \rfloor_{\zeta_m}$ and $\left\lfloor (1 \otimes \kappa)(\alpha_n)^{\frac{n}{m}} \right\rfloor_{(\zeta_n)^{\frac{n}{m}}}$.

Proposition 7.1.6. *Algorithm 11 is correct: it returns elements that define an embedding* $\phi : \mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$.

Proof. The ideas of the proof are the same as the ones found in the proof of Proposition 5.2.13, which were already present in the simpler case of Proposition 5.2.2, where all the roots of unity are in $\mathbf{k}$. By Proposition 7.1.3, let

$$\Phi = \phi \otimes \iota_{m,n}$$

be a Kummer embedding from A_m into A_n. Let $\alpha_m \in A_m$ a solution of (H90) for ζ_m with Kummer constant c_m, and let $\alpha_n \in A_n$ a solution of (H90) for ζ_n with Kummer constant c_n. By Proposition 7.1.4, there is a solution $\hat{\alpha} \in A_n$ of (H90) for $(\zeta_n)^{n/m}$ that satisfies $\hat{\alpha}^m = \iota_{m,n}(c_m)$ and such that $\Phi(\alpha_m) = \hat{\alpha}$. Now, we also have that $(\alpha_n)^{n/m}$ is a solution of (H90) for $(\zeta_n)^{n/m}$, and the solutions of (H90) for $(\zeta_n)^{n/m}$ form a one-dimensional $1 \otimes \mathbb{F}_p(\zeta_n)$-vector space, so there exists a constant $\lambda \in \mathbb{F}_p(\zeta_n)$ such that

$$\hat{\alpha} = (1 \otimes \lambda)(\alpha_n)^{n/m}.$$

We conclude that

$$\frac{\iota_{m,n}(c_m)}{c_n} = \lambda^m$$

is a m-th root in $\mathbb{F}_p(\zeta_n)$. If we let

$$\kappa = (\zeta_n)^{\frac{in}{m}}\lambda,$$

for some integer i, be a m-th root of $\frac{\iota_{m,n}(c_m)}{c_n}$, it follows that

$$\tilde{\alpha} = (1 \otimes \kappa)(\alpha_n)^{n/m} = (1 \otimes (\zeta_n)^{\frac{in}{m}})\hat{\alpha}$$

is a solution of (H90) for $(\zeta_n)^{n/m}$ that satisfies $\tilde{\alpha}^m = \iota_{m,n}(c_m)$. By Corollary 7.1.5, we know that the assignation

$$\lfloor \alpha_m \rfloor_{\zeta_m} \mapsto \lfloor \tilde{\alpha} \rfloor_{(\zeta_n)^{n/m}}$$

defines an embedding from $\mathbb{F}_{p^m}$ into $\mathbb{F}_{p^n}$. □

Remark 7.1.7. Taking the same notations as the one in the proof of Proposition 7.1.6, we see that the embedding returned by the algorithm is $\sigma^i \circ \phi$. Indeed, we have

$$\begin{aligned}\tilde{\alpha} &= (1 \otimes (\zeta_n)^{\frac{in}{m}})\hat{\alpha} \\ &= (\sigma \otimes 1)^i(\hat{\alpha}) \\ &= (\sigma^i \otimes 1)(\hat{\alpha}).\end{aligned}$$

If we let $x_0 = \lfloor \hat{\alpha} \rfloor_{(\zeta_n)^{\frac{n}{m}}}$, we wee that the returned embedding is defined by the assignation

$$\lfloor \alpha_m \rfloor_{\zeta_m} \mapsto \sigma^i(x_0)$$

while ϕ is defined by

$$\lfloor \alpha_m \rfloor_{\zeta_m} \mapsto x_0.$$

7.2 Standard solution of Hilbert 90

7.2.1 Complete algebras and standardization

We discussed in Section 7.1.1 the obstacles when constructing a lattice of compatibly embedded finite fields using the Lenstra-Allombert algorithm. The first one is that we need to have compatibility between the roots of unity that we use, and that can be hard to obtain in practice. We solved that problem by assuming the availability of a cyclotomic lattice $\mathcal{S}^I$, and we proved that all the results concerning the Lenstra-Allombert algorithm can be expressed in that setting in Section 7.1.3. Now, we can obtain compatibility by replacing the naive embedding algorithm by the Lenstra-Allombert algorithm in the Bosma-Canon-Steel framework. This solution immediately gives a compatible lattice of embedded finite fields. Still, among the sub-goals presented in Section 6.1 that such a lattice may achieve, there are two of them on which we would like to improve.

Uniqueness: the element $\lfloor \lambda_m \rfloor_{\zeta_m}$ is a generator of $\mathbb{F}_{p^m}$, or equivalently, it provides an irreducible polynomial in $\mathbf{k}[X]$ of degree m. However this polynomials depends on the choice of α_l, because it depends on the Kummer constant c_l of α_l by Proposition 5.2.12, thus there is no uniqueness.

Compatibility: the embedding of finite fields $\phi : \mathbb{F}_{p^m} \to \mathbb{F}_{p^n}$ depends on the choice of the constant κ, which itself depends on the choice of the solutions α_m and α_n of (H90), and also of the choice of a m-th root of unity. In order to achieve compatibility, we must keep track of the constants κ for each embedding computation

$$\mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$$

in the lattice, which grow quadratically with the number of fields, because of the common subfield compatibility condition in the Bosma-Canon-Steel framework.

In this section and in Section 7.3, we will see how to choose special solutions of (H90), in order to manage these constants κ. Our dream would be to be able to choose the solutions of (H90) in a way that makes the constants trivial, *i.e.*

$$\kappa_{\mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}} = 1.$$

From Algorithm 11, we see that the constant $\kappa_{\mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}}$ is a m-th root of the quotient

$$\frac{\iota_{m,n}(c_m)}{c_n},$$

thus the condition $\kappa_{\mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}} = 1$ implies

$$\iota_{m,n}(c_m) = c_n,$$

which in turn implies that c_n belongs to the subset

$$\mathbb{F}_p((\zeta_n)^{\frac{n}{m}}) \subseteq \mathbb{F}_p(\zeta_n).$$

This could possibly fail if the Kummer algebras A_m and A_n are of distinct level, *i.e.* if their field of scalars are different. This motivates the study of Kummer algebras of a given level, and the introduction of the notion of *complete* algebra.

Definition 7.2.1 (Complete Kummer algebra)**.** A Kummer algebra is *complete* if it is of the largest degree for a given level.

Therefore, the complete Kummer algebra of level a is the Kummer algebra

$$\begin{aligned} A_{p^a-1} &= \mathbb{F}_{p^{p^a-1}} \otimes \mathbb{F}_p(\zeta_{p^a-1}) \\ &= \mathbb{F}_{p^{p^a-1}} \otimes \mathbb{F}_{p^a} \end{aligned}$$

with field of scalars

$$\mathbb{F}_{p^a} \cong \mathbb{F}_p(\zeta_{p^a-1}).$$

given by the element ζ_{p^a-1} in the cyclotomic lattice $\mathcal{S}^I$. In fact these algebras have an interesting property.

Lemma 7.2.2. *All nonzero solutions $\alpha_{p^a-1} \in A_{p^a-1}$ of (H90) for ζ_{p^a-1} have the same Kummer constant*

$$c_{p^a-1} = (\zeta_{p^a-1})^a.$$

Proof. Let $\alpha_{p^a-1} \in A_{p^a-1}$ a solution of (H90) for ζ_{p^a-1}. For all $\beta \in A_{p^a-1}$, we have $\beta^p = \sigma \otimes \sigma(\beta) = \beta^p$, so we also obtain

$$(\alpha_{p^a-1})^{p^a} = (\sigma^a \otimes \sigma^a)(\alpha_{p^a-1}).$$

Now, we know that σ^a is the identity on $\mathbb{F}_{p^a}$, hence we have

$$\begin{aligned}(\alpha_{p^a-1})^{p^a} &= (\sigma^a \otimes 1)(\alpha_{p^a-1}) \\ &= (1 \otimes \zeta_{p^a-1})^a \alpha_{p^a-1}.\end{aligned}$$

Since α_{p^a-1} is invertible by Lemma 5.2.8, we obtain that

$$(\alpha_{p^a-1})^{p^a-1} = 1 \otimes c_{p^a-1} = (1 \otimes \zeta_{p^a-1})^a$$

and it follows that

$$c_{p^a-1} = (\zeta_{p^a-1})^a.$$

□

Now this result is very important because we know that the degree p^a-1 irreducible polynomial in $\mathbf{k}[X]$ derived from the solution α_{p^a-1}, *i.e.* the minimal polynomial of the element $\lfloor \alpha_{p^a-1} \rfloor_{\zeta_{p^a-1}}$, only depends on the Kummer constant c_{p^a-1} of α_{p^a-1}, thus it means that all solutions give the same polynomial. This will be the central idea behind the notion of *standard* elements.

Definition 7.2.3 (Standard Kummer constant)**.** Let m be an integer prime to p. We define the *standard Kummer constant* of order m as

$$\mathbf{c_m} = (\iota_{m,p^a-1})^{-1}((\zeta_{p^a-1})^a) \in \mathbb{F}_p(\zeta_m),$$

where $a = \nu(m)$ is the level of the Kummer algebra A_m.

Remark 7.2.4. Since A_m is of level a, we have

$$\mathbb{F}_p(\zeta_m) \cong \mathbb{F}_p(\zeta_{p^a-1}),$$

hence the map ι_{m,p^a-1} is an isomorphism and $\mathbf{c_m}$ is well-defined.

Definition 7.2.5 (Standard solution)**.** Let m be an integer prime to p. We say that a solution $\alpha_m \in A_m$ is *standard* if its Kummer constant is standard, *i.e.* if we have

$$(\alpha_m)^m = 1 \otimes \mathbf{c_m}.$$

Definition 7.2.6 (Decorated Kummer algebra)**.** Let m be an integer prime to p. We define a *decorated Kummer algebra* as a pair

$$(A_m, \alpha_m),$$

where α_m is a standard solution of (H90) for ζ_m.

It follows from Lemma 7.2.1 that all nonzero solutions of (H90) in a complete algebra are standard. This is no longer the case in a non complete Kummer algebra, but we can still find standard solutions.

Proposition 7.2.7. *Let m be an integer prime to p. Then A_m can be decorated,* i.e. *it admits a standard solution α_m. Moreover, this solution α_m is unique up to a m-th root of unity.*

Proof. Let $a = \nu(m)$ be the level of A_m and let $\alpha'_m \in A_m$ be a nonzero solution of (H90) for ζ_m. Let also α_{p^a-1} be a nonzero solution of (H90) for ζ_{p^a-1}, then it is standard by Lemma 7.2.1. The element

$$(\alpha_{p^a-1})^{\frac{p^a-1}{m}}$$

is a solution of (H90) for

$$\iota_{m,p^a-1}(\zeta_m) = (\zeta_{p^a-1})^{\frac{p^a-1}{m}}.$$

Now let

$$\Phi : A_m \hookrightarrow A_{p^a-1}$$

be a Kummer embedding and let

$$\hat{\alpha} = \Phi(\alpha'_m),$$

then $\hat{\alpha}$ is also a solution of (H90) for $\iota_{m,p^a-1}(\zeta_m)$ and thus there exists a scalar $\lambda \in \mathbb{F}_p(\zeta_{p^a-1})$ such that

$$(\alpha_{p^a-1})^{\frac{p^a-1}{m}} = (1 \otimes \lambda)\hat{\alpha}.$$

If we let

$$\tilde{\lambda} = \iota^{-1}_{m,p^a-1}(\lambda),$$

we obtain

$$\begin{aligned}(\alpha_{p^a-1})^{\frac{p^a-1}{m}} &= (1 \otimes \lambda)\Phi(\alpha'_m) \\ &= \Phi((1 \otimes \tilde{\lambda})\alpha'_m).\end{aligned}$$

If we set

$$\alpha_m = (1 \otimes \tilde{\lambda})\alpha'_m,$$

it follows that

$$\begin{aligned}\Phi((\alpha_m)^m) &= (\alpha_{p^a-1})^{p^a-1} \\ &= 1 \otimes (\zeta_{p^a-1})^a,\end{aligned}$$

therefore

$$\begin{aligned}(\alpha_m)^m &= 1 \otimes \iota^{-1}_{m,p^a-1}((\zeta_{p^a-1})^a) \\ &= 1 \otimes \mathbf{c_m}\end{aligned}$$

and α_m is standard. If $\beta \in A_m$ is another solution of (H90) for ζ_m, then there exists a scalar $\mu \in \mathbb{F}_p(\zeta_m)$ such that

$$\alpha_m = (1 \otimes \mu)\beta,$$

but then we obtain

$$(\alpha_m)^m = (1 \otimes \mu^m)\beta^m.$$

As a consequence, β is standard if and only if $\mu^m = 1$ and the standards solutions are the

$$(1 \otimes \zeta_m^u)\alpha_m$$

for $0 \leq u \leq m-1$. □

From Proposition 7.2.7, it follows that for any integer m prime to p, we can define $\mathbb{F}_{p^m}$ in a standard way.

$x + 1$
$x^3 + x + 1$
$x^5 + x^3 + 1$
$x^7 + x + 1$
$x^9 + x^7 + x^4 + x^2 + 1$
$x^{11} + x^8 + x^7 + x^6 + x^2 + x + 1$
$x^{13} + x^{10} + x^5 + x^3 + 1$
$x^{15} + x + 1$
$x^{17} + x^{11} + x^{10} + x^8 + x^7 + x^6 + x^4 + x^3 + x^2 + x + 1$
$x^{19} + x^{17} + x^{16} + x^{15} + x^{14} + x^{13} + x^{12} + x^8 + x^7 + x^6 + x^5 + x^3 + 1$

Table 7.1: The first ten standard polynomials derived from Conway polynomials for $p = 2$.

Definition 7.2.8 (Standard generating element)**.** A generating element $x \in \mathbb{F}_{p^m}$ is called *standard* if it is of the form

$$x = \lfloor \alpha_m \rfloor_{\zeta_m}$$

for $\alpha_m \in A_m$ a standard solution of (H90).

Definition 7.2.9 (Standard defining polynomial)**.** The *standard defining polynomial* P_m for $\mathbb{F}_{p^m}$ is the minimal polynomial over $\mathbf{k}$ of a standard generating element of $\mathbb{F}_{p^m}$.

Definition 7.2.10 (Decorated finite field)**.** Let $m \in \mathbb{N}$ an integer prime to p. A *decorated finite field* is a pair $(\mathbb{F}_{p^m}, s_m)$ where $s_m \in \mathbb{F}_{p^m}$ is a standard generating element.

Remark 7.2.11. By Remark 5.2.11, we can recover α_m the standard solution of (H90) from the standard generating element s_m, thus decorated Kummer algebras can be recovered from decorated finite fields.

By Proposition 5.2.12, we know that P_m is entirely determined by $\mathbf{c_m}$, and thus by the cyclotomic lattice $\mathcal{S}^I$ up to order $p^a - 1$, with $a = \nu(m)$ the level of m. This helps to achieve the *uniqueness* sub-goal of our lattice, because once the cyclotomic lattice has been chosen, the standard defining polynomials are unique. As an example, we give in Table 7.1 the first ten standard defining polynomials induced by the cyclotomic lattice given by the Conway polynomials with $p = 2$. A small variation of the Lenstra-Allombert algorithm, given by Algorithm 12, allows us to compute all these standard elements.

Algorithm 12 (Decoration – Standardization)

Input: $\mathbb{F}_{p^m}$, for m prime to p, and $\mathcal{S}^I$ a cyclotomic lattice.
Output: (A_m, α_m) decorated, P_m standard irreducible polynomial of degree m, and $s \in \mathbb{F}_{p^m}$ standard generating element inducing $\mathbb{F}_{p^m} \simeq \mathbb{F}_p[T]/(P_m)$.

1: Compute the Kummer algebra A_m.
2: Compute $\mathbf{c_m} = (\iota_{m,p^a-1})^{-1}((\zeta_{p^a-1})^a) \in \mathbb{F}_p(\zeta_m)$.
3: Find $\alpha'_m \in A_m$ a nonzero solution of (H90) for ζ_m.
4: Compute its Kummer constant: $(\alpha'_m)^m = 1 \otimes c'_m$.
5: Compute κ a m-th root of $\mathbf{c_m}/c'_m$.
6: Set $\alpha_m = (1 \otimes \kappa)\alpha'_m$.
7: Compute P_m the minimal polynomial of $\lfloor \alpha_m \rfloor_{\zeta_m}$ over $\mathbf{k}$.
8: Return (A_m, α_m), P_m, and $\lfloor \alpha_m \rfloor_{\zeta_m}$.

Proposition 7.2.12. *Algorithm 12 is correct,* i.e. *the computed element α_m is indeed standard.*

Proof. By Proposition 7.2.7, we know that there exists a standard solution of (H90) for ζ_m, let $\boldsymbol{\alpha} \in A_m$ be such a solution. Let $\alpha'_m \in A_m$ be any nonzero solution of (H90) for ζ_m and let c'_m be its Kummer constant. Then we know that there exists $\lambda \in \mathbb{F}_p(\zeta_m)$ a scalar such that

$$\boldsymbol{\alpha} = (1 \otimes \lambda)\alpha'_m.$$

It follows that

$$\mathbf{c_m} = \lambda^m c'_m,$$

thus

$$\mathbf{c_m}/c'_m$$

is indeed a m-th power. If we let κ be a m-th root of $\mathbf{c_m}/c'_m$ and if we set

$$\alpha_m = (1 \otimes \kappa)\alpha'_m,$$

we obtain

$$(\alpha_m)^m = 1 \otimes \mathbf{c_m}.$$

We have thus proven that α_m is a standard solution of (H90) for ζ_m. By definition, $\lfloor \alpha_m \rfloor_{\zeta_m}$ is then a standard generating element of $\mathbb{F}_{p^m}$ and its minimal polynomial P_m is a standard defining polynomial. □

7.2.2 Towards standard embeddings

Now that we have found a way to compute standard solutions of (H90), and to deduce standard generating elements to define our finite fields, it is natural to work towards the definition of compatible embeddings between these finite fields that are also standard. Our goal is also to simplify the storage of the constants κ involved in the computation of such embeddings.

Proposition 7.2.13. *Let $m \mid n$ be two integers prime to p and such that*

$$\nu(m) = \nu(n),$$

i.e. *the (decorated) Kummer algebras (A_m, α_m) and (A_n, α_n) have the same level. Then there is a unique Kummer embedding*

$$\boldsymbol{\Phi_{m,n}} : A_m \hookrightarrow A_n$$

such that

$$\boldsymbol{\Phi_{m,n}}(\alpha_m) = (\alpha_n)^{\frac{n}{m}}.$$

Proof. The element $\hat{\alpha} = (\alpha_n)^{\frac{n}{m}}$ is a solution of (H90) for $(\zeta_n)^{\frac{n}{m}} = \iota_{m,n}(\zeta_m)$. We also know that the Kummer constant of α_m is

$$\mathbf{c_m} = \iota^{-1}_{m,p^a-1}((\zeta_{p^a-1})^a)$$

and the Kummer constant of α_n is

$$\mathbf{c_n} = \iota^{-1}_{n,p^a-1}((\zeta_{p^a-1})^a).$$

The embeddings ι are compatible by definition, thus we have

$$\iota_{m,p^a-1} = \iota_{n,p^a-1} \circ \iota_{m,n}$$

and

$$\iota_{n,p^a-1}^{-1} = \iota_{m,n} \circ \iota_{m,p^a-1}^{-1}.$$

It follows that

$$\mathbf{c_n} = \iota_{m,n}(\mathbf{c_m}).$$

By Proposition 7.1.4, there exists a unique Kummer embedding $\boldsymbol{\Phi_{m,n}}$ such that

$$\boldsymbol{\Phi_{m,n}}(\alpha_m) = \hat{\alpha} = (\alpha_n)^{\frac{n}{m}}.$$

□

Proposition 7.2.13 guarantees that decorated Kummer algebras $(A_m, \alpha_m), (A_n, \alpha_n)$ of the same level are *power-compatible*, *i.e.* we can describe a unique Kummer embedding by

$$\alpha_m \mapsto (\alpha_n)^{\frac{n}{m}}.$$

Therefore, there is no constant κ to store in this case because we are in the trivial case $\kappa = 1$, which was our goal. However, we see that power compatibility implies

$$\mathbf{c_n} = \iota_{m,n}(\mathbf{c_m}),$$

hence implies that the decorated Kummer algebras share the same level. If the Kummer algebras do not share the same level, we can ask for *norm-compatibility* instead of power-compatibility, at least between two *complete* decorated Kummer algebras. Let us first describe what "norm" we will be using. Let A_n be a Kummer algebra of level $\nu(n) = b$, so that

$$A_n = \mathbb{F}_{p^n} \otimes \mathbb{F}_{p^b} \cong \mathbb{F}_{p^n} \otimes \mathbb{F}_p(\zeta_{p^b-1}),$$

where the isomorphism is given by $1 \otimes \iota_{n,p^b-1}$. Then, if $a \mid b$ is another integer, the subalgebra of A_n invariant under $1 \otimes \sigma^a$ is identified (by the same isomorphism) with

$$(A_n)^{1\otimes\sigma^a} \cong \mathbb{F}_{p^n} \otimes \mathbb{F}_p((\zeta_{p^b-1})^{\frac{p^b-1}{p^a-1}})$$

where

$$(\zeta_{p^b-1})^{\frac{p^b-1}{p^a-1}} = N_{\mathbb{F}_{p^n}/\mathbb{F}_{p^a}}(\zeta_{p^b-1}) = \iota_{p^a-1,p^b-1}(\zeta_{p^a-1})$$

and where $N_{\mathbb{F}_{p^n}/\mathbb{F}_{p^a}}$ is the relative norm of the finite field extension

$$\mathbb{F}_{p^b}/\mathbb{F}_{p^a}.$$

Definition 7.2.14 (Scalar norm operator)**.** Let n an integer prime to p and A_n a Kummer algebra of level $\nu(n) = c$. Let a, b two integers such that $a \mid b \mid c$, then we define the *scalar norm operator* as

$$\begin{array}{rccc} \mathcal{N}_{b/a,\,A_n}: & (A_n)^{1\otimes\sigma^b} & \to & (A_n)^{1\otimes\sigma^a} \\ & \gamma & \mapsto & \prod_{0\le j\le \frac{a}{b}}(1\otimes\alpha^{ja})(\gamma) \end{array}$$

This operator is well-defined, as the elements in the image of $\mathcal{N}_{b/a,\,A_n}$ are indeed invariant under $1 \otimes \sigma^a$. We often omit A_n and only write $\mathcal{N}_{b/a}$, as the ambient space A_n is implicit. By construction, $\mathcal{N}_{b/a}$ acts on the scalar field $1 \otimes \mathbb{F}_{p^b}^\times$ as the usual norm $1 \otimes N_{\mathbb{F}_{p^b}/\mathbb{F}_{p^a}}$. Scalar norms are *multiplicative*, *i.e.* for any $\gamma, \gamma' \in A_n$, we have

$$\mathcal{N}_{b/a}(\gamma\gamma') = \mathcal{N}_{b/a}(\gamma)\mathcal{N}_{b/a}(\gamma'),$$

they are *transitive*, *i.e.* for any $a \mid b \mid c$, we have

$$\mathcal{N}_{c/a} = \mathcal{N}_{b/a} \circ \mathcal{N}_{c/b},$$

and they commute with $\sigma \otimes 1$. All these nice properties make the scalar norm an excellent candidate to generalize the powering used to define standard embeddings between decorated Kummer algebras of same level.

Proposition 7.2.15. *Let $a \mid b$ be two integers prime to p and let $(A_{p^a-1}, \alpha_{p^a-1})$, $(A_{p^b-1}, \alpha_{p^b-1})$ two decorated complete Kummer algebras of levels a and b respectively. Then there is a unique Kummer embedding (that we again call* standard*)*

$$\boldsymbol{\Phi_{p^a-1,\,p^b-1}} : A_{p^a-1} \hookrightarrow A_{p^b-1}$$

such that

$$\boldsymbol{\Phi_{p^a-1,\,p^b-1}}(\alpha_{p^a-1}) = \mathcal{N}_{b/a}(\alpha_{p^b-1}).$$

Proof. Let $\hat{\alpha} = \mathcal{N}_{b/a}(\alpha_{p^b-1})$, by the properties of the scalar norm, we have

$$\begin{aligned}(\sigma \otimes 1)(\hat{\alpha}) &= (\sigma \otimes 1)(\mathcal{N}_{b/a}(\alpha_{p^b-1})) \\ &= \mathcal{N}_{b/a}((\sigma \otimes 1)(\alpha_{p^b-1})) \\ &= \mathcal{N}_{b/a}((1 \otimes \zeta_{p^b-1})\alpha_{p^b-1}) \\ &= (1 \otimes (\zeta_{p^b-1})^{\frac{p^b-1}{p^a-1}})\mathcal{N}_{b/a}(\alpha_{p^b-1}),\end{aligned}$$

therefore $\hat{\alpha}$ is a solution of (H90) for $(\zeta_{p^b-1})^{\frac{p^b-1}{p^a-1}}$. We also have

$$\begin{aligned}\hat{\alpha}^{p^a} &= (\sigma^a \otimes \sigma^a)(\hat{\alpha}) \\ &= (\sigma^a \otimes \sigma^a)(\mathcal{N}_{b/a}(\alpha_{p^b-1})) \\ &= (\sigma^a \otimes 1)(\mathcal{N}_{b/a}(\alpha_{p^b-1})) \\ &= \mathcal{N}_{b/a}((\sigma^a \otimes 1)(\alpha_{p^b-1})) \\ &= \mathcal{N}_{b/a}((1 \otimes (\zeta_{p^b-1})^a)(\alpha_{p^b-1})) \\ &= (1 \otimes \iota_{p^a-1,\,p^b-1}(\zeta_{p^a-1})^a)\mathcal{N}_{b/a}(\alpha_{p^b-1}),\end{aligned}$$

thus $\hat{\alpha}$ satisfies

$$\hat{\alpha}^{p^a-1} = 1 \otimes \iota_{p^a-1,\,p^b-1}(\mathbf{c_{p^a-1}}),$$

and by Proposition 7.1.4 there is a unique embedding such that

$$\boldsymbol{\Phi_{p^a-1,\,p^b-1}}(\alpha_{p^b-1}) = \hat{\alpha}.$$

□

7.3 Standard embeddings

From Proposition 7.2.13, we learned how to construct a standard Kummer embedding between two decorated Kummer algebras sharing the same level, using power compatibility. From Proposition 7.2.15 we learned how to construct a standard Kummer embedding between two

decorated complete Kummer algebras, using norm compatibility. Our goal is now to use both results together in order to construct a standard Kummer embedding between any two decorated Kummer algebras. Consider the general case where we have two integers $m \mid n$ not divisible by p, set $a = \nu(m)$, $b = \nu(n)$, and consider the diagram

$$\begin{array}{ccc} (A_{p^a-1}, \alpha_{p^a-1}) & \xrightarrow{\boldsymbol{\Phi_{p^a-1,\,p^b-1}}} & (A_{p^b-1}, \alpha_{p^b-1}) \\ {\scriptstyle \boldsymbol{\Phi_{m,\,p^a-1}}}\uparrow & & \uparrow{\scriptstyle \boldsymbol{\Phi_{n,\,p^b-1}}} \\ (A_m, \alpha_m) & & (A_n, \alpha_n) \end{array}$$

of standard embeddings of decorated algebras.

Lemma 7.3.1. *In this setting, there exists a unique Kummer embedding*

$$\boldsymbol{\Phi_{m,n}} : A_m \hookrightarrow A_n$$

that makes the diagram commute. We call this embedding the standard Kummer embedding *from* A_m *to* A_n.

Proof. Let

$$\tilde{\alpha} = \boldsymbol{\Phi_{p^a-1,\,p^b-1}}(\boldsymbol{\Phi_{m,\,p^a-1}}(\alpha_l)) \in A_{p^b-1}.$$

The element $\tilde{\alpha}$ is fixed by both $\sigma^m \otimes 1$ and $1 \otimes \sigma^a$, because α_m is, and Kummer embeddings commute with both $\sigma \otimes 1$ and $1 \otimes \sigma$. Therefore, $\tilde{\alpha}$ is also fixed by $\sigma^n \otimes 1$ and $1 \otimes \sigma^b$, and thus is in the image of A_n by $\boldsymbol{\Phi_{n,\,p^b-1}}$. We then let

$$\hat{\alpha} = (\boldsymbol{\Phi_{n,\,p^b-1}})^{-1}(\tilde{\alpha}) \in A_n.$$

By construction, the element $\tilde{\alpha}$ is a solution of (H90) for

$$\iota_{m,\,p^b-1}(\zeta_m) = (\zeta_{p^b-1})^{\frac{p^b-1}{m}}$$

that satisfies

$$\tilde{\alpha}^m = 1 \otimes \iota_{m,\,p^b-1}(\mathbf{c_m})$$

It thus follows that $\hat{\alpha}$ is a solution of (H90) for

$$\iota^{-1}_{n,\,p^b-1}(\iota_{m,\,p^b-1}(\zeta_m)) = \iota_{m\,n}(\zeta_m) = (\zeta_n)^{\frac{n}{m}}$$

that also satisfies

$$\hat{\alpha}^m = 1 \otimes \iota_{m,n}(\mathbf{c_m}).$$

By Proposition 7.1.4, we then have a unique Kummer embedding such that

$$\boldsymbol{\Phi_{m,n}}(\alpha_m) = \hat{\alpha},$$

and that concludes the proof. □

Definition 7.3.2 (Standard embedding)**.** Let $m \mid n$ two integers prime to p. By Proposition 7.1.3, there exists a unique finite field embedding

$$\boldsymbol{\phi_{m,n}} : \mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$$

such that the standard Kummer embedding $\boldsymbol{\Phi_{m,n}}$ satisfies

$$\boldsymbol{\Phi_{m,n}} = \boldsymbol{\phi_{m,n}} \otimes \iota_{m,n}.$$

This finite field embedding $\boldsymbol{\phi_{m,n}}$ is called the *standard embedding* from $\mathbb{F}_{p^m}$ to $\mathbb{F}_{p^n}$.

The existence result of Lemma 7.3.1 is "constructive", but requires to compute the complete Kummer algebra A_{p^a-1}, that can be very large, thus it is impractical. However, one should be able to write

$$\hat{\alpha} = (1 \otimes \kappa)(\alpha_n)^{\frac{n}{m}}$$

for some constant $\kappa \in \mathbb{F}_p(\zeta_n)$, since both $\hat{\alpha}$ and $(\alpha_n)^{\frac{n}{m}}$ are solutions of (H90) for $(\zeta_n)^{\frac{n}{m}}$. We proved in Lemma 7.3.1 that there was a unique, *standard*, Kummer embedding corresponding to this solution $\hat{\alpha}$, thus there should also be a unique constant κ corresponding to this embedding. Our aim is now to give an explicit expression of κ, so that we do not have to compute the complete algebra A_{p^b-1} to obtain the Kummer embedding

$$\boldsymbol{\Phi_{m,n}}.$$

We start with the simpler case of complete algebras nonetheless, *i.e.* we study the constant κ when $m = p^a - 1$ and $n = p^b - 1$.

Proposition 7.3.3. *In the complete algebra A_{p^b-1} we have*

$$(\alpha_{p^b-1})^{\frac{p^b-1}{p^a-1}} = (1 \otimes \zeta_{p^b-1})^{\frac{(b-a)p^{b+a} - bp^b + ap^a}{(p^a-1)^2}} \mathcal{N}_{b/a}(\alpha_{p^b-1}).$$

Proof. Using the fact that $(\sigma \otimes \sigma)(\beta) = \beta^p$ for any $\beta \in A_{p^b-1}$, and then (H90), we get:

$$\begin{aligned}
\frac{(\alpha_{p^b-1})^{\frac{p^b-1}{p^a-1}}}{\mathcal{N}_{b/a}(\alpha_{p^b-1})} &= \prod_{0 \le j < \frac{b}{a}} \frac{(\sigma^{ja} \otimes \sigma^{ja})(\alpha_{p^b-1})}{(1 \otimes \sigma^{ja})(\alpha_{p^b-1})} \\
&= \prod_{0 \le j < \frac{b}{a}} (1 \otimes \sigma^{ja}) \left(\frac{(\sigma^{ja} \otimes 1)(\alpha_{p^b-1})}{\alpha_{p^b-1}} \right) \\
&= \prod_{0 \le j < \frac{b}{a}} (1 \otimes \sigma^{ja})(1 \otimes \zeta_{p^b-1})^{ja} \\
&= (1 \otimes \zeta_{p^b-1})^{\sum_{0 \le j < \frac{b}{a}} jap^{ja}}.
\end{aligned}$$

We conclude thanks to the identity

$$\sum_{0 \le j < n} jT^j = T \frac{d}{dT} \left(\frac{T^n - 1}{T - 1} \right) = \frac{(n-1)T^{n+1} - nT^n + T}{(T-1)^2}.$$

□

From Proposition 7.3.3, we can deduce the constant κ involved in the computation of $\boldsymbol{\Phi_{p^a-1,p^b-1}}$. The results also extends to non complete algebras.

Corollary 7.3.4. *Let (A_m, α_n) and (A_n, α_n) be two decorated Kummer algebras of respective degrees $m \mid n$ prime to p and respective levels a and b. Then the standard embedding*

$$\boldsymbol{\Phi_{m,n}} : A_m \hookrightarrow A_n$$

is defined by the assignation

$$\alpha_m \mapsto (1 \otimes \kappa_{m,n})(\alpha_n)^{\frac{n}{m}},$$

where

$$\kappa_{m,n} = \iota^{-1}_{n,p^b-1}(\zeta_{p^b-1})^{-\frac{(b-a)p^{b+a} - bp^b + ap^a}{(p^a-1)m}}.$$

Proof. Recall that $\boldsymbol{\Phi_{m,n}}$ is, by construction, the unique Kummer embedding that makes the following diagram commute.

$$\begin{array}{ccc} (A_{p^a-1}, \alpha_{p^a-1}) & \xrightarrow{\boldsymbol{\Phi_{p^a-1,p^b-1}}} & (A_{p^b-1}, \alpha_{p^b-1}) \\ {\scriptstyle \boldsymbol{\Phi_{m,p^a-1}}}\uparrow & & \uparrow{\scriptstyle \boldsymbol{\Phi_{n,p^b-1}}} \\ (A_m, \alpha_m) & & (A_n, \alpha_n) \end{array}$$

Therefore, if we set

$$\hat{\alpha} = (1 \otimes \kappa_{m,n})(\alpha_n)^{\frac{n}{m}},$$

it is sufficient, by Lemma 7.3.1, to prove that

$$\boldsymbol{\Phi_{p^a-1,p^b-1}}(\boldsymbol{\Phi_{m,p^b-1}})(\alpha_m) = \boldsymbol{\Phi_{n,p^b-1}}(\hat{\alpha}).$$

However, using Proposition 7.2.15 and Proposition 7.2.13, we know that

$$\boldsymbol{\Phi_{p^a-1,p^b-1}}(\boldsymbol{\Phi_{m,p^b-1}})(\alpha_m) = \mathcal{N}_{b/a}(\alpha_{p^b-1})^{\frac{p^a-1}{m}},$$

and we also know that

$$\boldsymbol{\Phi_{n,p^b-1}}((\alpha_m)^{\frac{n}{m}}) = (\alpha_{p^b-1})^{\frac{p^b-1}{m}}.$$

Therefore, using Proposition 7.3.3, we obtain

$$\begin{aligned} \boldsymbol{\Phi_{n,p^b-1}}(\hat{\alpha}) &= (\zeta_{p^b-1})^{-\frac{(b-a)p^{b+a}-bp^b+ap^a}{(p^a-1)m}}(\alpha_{p^b-1})^{\frac{p^b-1}{m}} \\ &= ((\zeta_{p^b-1})^{-\frac{(b-a)p^{b+a}-bp^b+ap^a}{(p^a-1)^2}}(\alpha_{p^b-1})^{\frac{p^b-1}{p^a-1}})^{\frac{p^a-1}{m}} \\ &= \mathcal{N}_{b/a}(\alpha_{p^b-1})^{\frac{p^a-1}{m}} \\ &= \boldsymbol{\Phi_{p^a-1,p^b-1}}(\boldsymbol{\Phi_{m,p^b-1}})(\alpha_m), \end{aligned}$$

which completes the proof. □

Proposition 7.3.5. *Standard Kummer embeddings are compatible: let $(A_l, \alpha_l), (A_m, \alpha_m)$ and (A_n, α_n) three decorated Kummer algebras with respective degrees*

$$l \mid m \mid n,$$

then the corresponding standard embeddings satisfy

$$\boldsymbol{\Phi_{l,n}} = \boldsymbol{\Phi_{m,n}} \circ \boldsymbol{\Phi_{l,m}}.$$

Proof. Corollary 7.3.4 gives us explicit formulas for the standard Kummer embeddings, thus we can check by direct computation that the embeddings are compatible. Let

$$a = \nu(l) \quad b = \nu(m) \quad c = \nu(n)$$

the respective levels of A_l, A_m and A_n. Then we have the corresponding commutative diagram where the arrows represent the standard Kummer embeddings.

$$\begin{array}{ccccc} A_{p^a-1} & \longrightarrow & A_{p^b-1} & \longrightarrow & A_{p^c-1} \\ \uparrow & & \uparrow & & \uparrow \\ A_l & \longrightarrow & A_m & \longrightarrow & A_n \end{array}$$

Using Lemma 7.3.1, it is sufficient to show that

$$\boldsymbol{\Phi_{l,n}}(\alpha_l)$$

and

$$(\boldsymbol{\Phi_{m,n}} \circ \boldsymbol{\Phi_{l,m}})(\alpha_l)$$

have the same image in A_{p^c-1} under $\boldsymbol{\Phi_{n,p^c-1}}$. However, it follows from the diagram that this common image is

$$\mathcal{N}_{c/a}(\alpha_{p^c-1})^{\frac{p^a-1}{l}}.$$

□

All these results ensure that Algorithm 13 is correct and provides embeddings that are compatible.

Algorithm 13 (Standard compatible embeddings)

Input: $\mathcal{S}^I$ a cyclotomic lattice, and $(\mathbb{F}_{p^m}, s_m)$, $(\mathbb{F}_{p^n}, s_n)$, decorated finite fields, for $m \mid n$ integers prime to p.

Output: $t \in \mathbb{F}_{p^n}$, such that the assignation $s_m \mapsto t$ defines a standard embedding $\boldsymbol{\phi_{m,n}} : \mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$, compatible with composition.

1: Compute the Kummer algebras A_m and A_n.

2: Recover α_m from s_m and α_n from s_n using Remark 5.2.11.

3: Compute $\kappa_{m,n} = (\iota_{n,p^b-1})^{-1}\left((\zeta_{p^b-1})^{-\frac{(b-a)p^{b+a}-bp^b+ap^a}{(p^a-1)m}}\right)$ where $a = \nu(m)$, $b = \nu(n)$.

4: Return $\left\lfloor (1 \otimes \kappa)(\alpha_n)^{\frac{n}{m}} \right\rfloor_{(\zeta_n)^{\frac{n}{m}}}$.

Proposition 7.3.6. *Standard finite field embeddings are compatible with composition: if* $(\mathbb{F}_{p^l}, s_l)$, $(\mathbb{F}_{p^m}, s_m)$, *and* $(\mathbb{F}_{p^n}, s_n)$ *are decorated finite fields with* $l \mid m \mid n$, *the corresponding standard embeddings satisfy* $\boldsymbol{\phi_{l,n}} = \boldsymbol{\phi_{m,n}} \circ \boldsymbol{\phi_{l,m}}$.

Proof. Proposition 7.3.5 ensures that standard Kummer embedding are compatible, and by Corollary 7.1.5 this also implies the compatibility of the standard embeddings between the finite fields. □

7.4 Implementation

In the preceding sections, we proposed a new method to construct a lattice of compatibly embedded finite fields and proved that our method was correct. However, the description of Kummer embedding was abstract, and many computational details were left unspecified. There are various ways in which the algorithms can be implemented, depending on how the finite fields are represented and how the cyclotomic lattice $\mathcal{S}^I$ is constructed. In order to demonstrate the praticality of this method, we implemented it in Nemo/Flint [H+16, Har10]. In this section, we present the experimental results and the complexity analysis, depending on the representation we used in our code.

7.4.1 Complexity analysis

In order to prove a bound on the complexity of our algorithms, we have to specify which representation we assume for our finite fields. A reasonable option, and also the one we use in practice, is to use Conway polynomials to represent the cyclotomic part of the lattice, *i.e.* the fields

$$\mathbb{F}_p(\zeta).$$

To do so, we use the Conway polynomials to represent the finite fields

$$\mathbb{F}_p(\zeta_{p^a-1})$$

and we deduce from them the smallest possible representation for any other field

$$\mathbb{F}_p(\zeta_m).$$

We first study the complexity of computing a standard solution of (H90).

Proposition 7.4.1. *Given a collection of Conway polynomials for* $\mathbf{k} = \mathbb{F}_p$*, of degree up to* d*, standard solutions* α_m *of* (H90) *can be computed for any* $m \mid (p^i - 1)$ *for any* $i \leq d$ *using*

$$O(M(m^2)\log(m) + M(m)\log(m)\log(p))$$

operations.

Proof. Let

$$a = \nu(m)$$

be the level of A_m, we take the a-th Conway polynomial from the collection and use it to define ζ_{p^a-1}. We have

$$a = \mathrm{ord}_{(\mathbb{Z}/m\mathbb{Z})^\times}(p)$$

where $\mathrm{ord}_{(\mathbb{Z}/m\mathbb{Z})^\times}$ is the multiplicative order in the group $(\mathbb{Z}/m\mathbb{Z})^\times$, and since $a \leq m-1$, we have $a = O(m)$, and then the cost of multiplication in

$$\mathbb{F}_p(\zeta_{p^a-1}) \cong \mathbb{F}_{p^a}$$

is bounded by $O(M(m))$. We can then compute

$$(\zeta_{p^a-1})^{\frac{p^a-1}{m}}$$

using $O(mM(m))$ operations in $\mathbf{k}$ with the square-and-multiply algorithm. Then, its minimal polynomial can be computed using $O(m^{\frac{\omega+1}{2}})$ operations with the algorithm presented in [Sho94]. The Kummer constant

$$\mathbf{c_{p^a-1}} = (\zeta_{p^a-1})^a$$

can then be computed using a negligible (logarithmic in a) number of operations in $\mathbf{k}$, and the Kummer constant

$$\mathbf{c_m} = \iota^{-1}_{m,\,p^a-1}(\mathbf{c_{p^a-1}})$$

can be computed using the algorithm presented in Section 5.3.3, that is also based on [Sho94], within the same complexity bound $O(m^{\frac{\omega+1}{2}})$. To construct the Kummer algebra

$$A_m = \mathbb{F}_{p^m} \otimes \mathbb{F}_p(\zeta_m),$$

we only need an irreducible polynomial over $\mathbf{k}$ of degree m, since we already have the minimal polynomial of ζ_m. Very efficient, quasi-optimal algorithms [BFSS06, CL13, DFDS13] exist to compute such polynomials, thus the cost is also negligible. We then represent the Kummer algebra as

$$\mathbb{F}_{p^m}[T]/(h(T))$$

where h is the minimal polynomial of ζ_m over $\mathbf{k}$, and where $\mathbb{F}_{p^m}$ is defined with the irreducible polynomial of degree m that we found. With the Kummer algebra constructed, we can compute a solution $\alpha_m \in A_m$ of (H90) for ζ_m at a cost of

$$O(M(m^2)\log(m) + M(m)\log(p))$$

operations in $\mathbf{k}$, as explained in Section 5.2.4. We can then compute the Kummer constant

$$c'_m = (\alpha'_m)^m$$

with $O(M(m^2)\log(m))$ operations in $\mathbf{k}$ via Kronecker substitution, and the m-th root extraction of

$$\mathbf{c_m}/c'_m = \kappa^m$$

costs $O(M(m)\log(m)\log(p))$ operations in $\mathbf{k}$, according to the analysis in [BDFD+17]. The standard solution

$$\alpha_m = (1 \otimes \kappa)\alpha'_m$$

is finally computed with a negligible number of operations. In conclusion, the two dominating steps are the m-th root extraction and the computation of the solution of (H90), hence the total complexity is

$$O(M(m^2)\log(m) + M(m)\log(m)\log(p)).$$

□

Proposition 7.4.2. *Under the same assumptions as in Proposition 7.4.1 and after the computation of two decorated algebras (A_m, α_m) and (A_n, α_n), a standard embedding of finite fields*

$$\mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$$

can be computed using

$$O(M(n^2)\log(n))$$

operations in $\mathbf{k}$.

Proof. The projection

$$x_m = \lfloor \alpha_m \rfloor_{\zeta_m}$$

comes for free because α_m is already represented in the $(\mathbb{F}_{p^m} \otimes 1)$-basis $(1 \otimes (\zeta_m)^j)_j$ of A_m, *i.e.* as

$$\alpha_m = \sum_{j=0}^{m-1} a_j \otimes (\zeta_m)^j,$$

and the minimal polynomial of x_m over $\mathbf{k}$ is computed using $O(m^{\frac{\omega+1}{2}})$ operations. We now need to compute the scalar

$$\kappa_{m,n} = \iota^{-1}_{n,p^b-1}(\zeta_{p^b-1})^{-\frac{(b-a)p^{b+a}-bp^b+ap^a}{(p^a-1)m}}.$$

which is done using $O(M(n)n)$ operations in $\mathbf{k}$, and the element

$$(\alpha_n)^{\frac{n}{m}},$$

which is done using $O(M(n^2)\log(n))$ operations. The only remaining step is to compute

$$x_n = \left\lfloor (1 \otimes \kappa_{m,n})(\alpha_n)^{\frac{n}{m}} \right\rfloor_{(\zeta_n)^{\frac{n}{m}}},$$

but it is not free this time, because

$$\hat{\alpha} = (1 \otimes \kappa_{m,n})(\alpha_n)^{\frac{n}{m}}$$

is represented in the basis $(1 \otimes (\zeta_n)^j)_j$, when we need a representation in the basis $(1 \otimes (\zeta_n)^{\frac{jn}{m}})_j$. A generic change of basis algorithm would be too expensive because we need to convert n elements from the field of scalar $\mathbb{F}_p(\zeta_n)$ to

$$\mathbb{F}_p((\zeta_n)^{\frac{n}{m}}) \cong \mathbb{F}_p(\zeta_m),$$

which costs $O(n^{\frac{\omega+3}{2}})$. Instead, we note that we only need

$$\lfloor \hat{\alpha} \rfloor_{(\zeta_n)^{\frac{n}{m}}},$$

thus we only need one coordinate in the basis $(1 \otimes (\zeta_n)^{\frac{jn}{m}})_{0 \leq j \leq m-1}$, and we proceed as follows. Let Tr denote the trace map of the extension

$$\mathbb{F}_p(\zeta_n)/\mathbb{F}_p((\zeta_n)^{\frac{n}{m}})$$

and let $u \in \mathbb{F}_p(\zeta_n)$ be an element such that $\mathrm{Tr}(u) = 1$. In this case, the embedding

$$\mathbb{F}_p((\zeta_n)^{\frac{n}{m}}) \hookrightarrow \mathbb{F}_p(\zeta_n)$$

is just the identity because we have

$$\mathbb{F}_p((\zeta_n)^{\frac{n}{m}}) \subseteq \mathbb{F}_p(\zeta_n).$$

Then, as explained in Section 5.3.2, the map

$$x \mapsto \mathrm{Tr}(ux)$$

is $\mathbb{F}_p((\zeta_n)^{\frac{n}{m}})$-linear and every element in $\mathbb{F}_p((\zeta_n)^{\frac{n}{m}})$ is fixed, thus the map agrees with the inverse of the embedding

$$\mathbb{F}_p((\zeta_n)^{\frac{n}{m}}) \hookrightarrow \mathbb{F}_p(\zeta_n)$$

on its image $\mathbb{F}_p((\zeta_n)^{\frac{n}{m}})$. We thus need to obtain the first coordinate of

$$\mathrm{Tr}(ux)$$

in the power basis of $(\zeta_n)^{\frac{n}{m}}$, that we denote by

$$\lfloor \mathrm{Tr}(ux) \rfloor_{(\zeta_n)^{\frac{n}{m}}}.$$

In the end, we need to evaluate the map

$$x \mapsto \lfloor \mathrm{Tr}(ux) \rfloor_{(\zeta_n)^{\frac{n}{m}}}$$

for many values $x \in \mathbb{F}_p(\zeta_n)$, but this map is a $\mathbf{k}$-linear form, hence we can precompute its vector on the power basis of ζ_n. Let h_n, h_m be the minimal polynomials of ζ_n and $(\zeta_n)^{\frac{n}{m}}$, and let b, a be their degrees. Let h_0 be the constant coefficient of h_m, and let

$$\tau = -\frac{h_0}{(\zeta_n)^{\frac{n}{m}}} \frac{h_n'(\zeta_n)}{h_m'((\zeta_n)^{\frac{n}{m}})} \in \mathbb{F}_p(\zeta_n),$$

direct calculation shows that

$$\sum_{i=0}^{b-1} \left\lfloor \mathrm{Tr}(\zeta_n^i) \right\rfloor_{(\zeta_n)^{\frac{n}{m}}} Z^i = \frac{\tau(Z^{-1})}{Z h_n(Z^{-1})} \mod Z^b,$$

where by $\tau(Z)$ we mean $\tau \in \mathbb{F}_p(\zeta_n)$ seen as a polynomial in ζ_n. Hence, we can compute the vector of the linear form $x \mapsto \left\lfloor \mathrm{Tr}(x) \right\rfloor_{(\zeta_n)^{\frac{n}{m}}}$ using only basic polynomial arithmetic and modular composition, *i.e.* in $O(n^{(\omega+1)/2})$ operations. Finally, we compute

$$(1 \otimes u)\hat{\alpha} = (1 \otimes \kappa_{m,n} u)(\alpha_n)^{\frac{n}{m}},$$

we see it as a polynomial with coefficients in $\mathbb{F}_p(\zeta_n)$, and we apply the map $\left\lfloor \mathrm{Tr}(x) \right\rfloor_{(\zeta_n)^{\frac{n}{m}}}$ to each coefficient to recover x_n. This costs $O(nM(n))$ operations. □

In terms of memory complexity, we remark that storing the standard solution of (H90) α_m costs $O(m^2)$ elements in $\mathbf{k}$, but thanks to Remark 5.2.11, we only have to store

$$\left\lfloor \alpha_m \right\rfloor_{\zeta_m},$$

hence we need only $O(m)$ field elements.

7.4.2 Experimental results

We implemented our lattice of compatibly embedded finite fields in the computer algebra system Nemo [H+16], written in the Julia [Jul] programming language, and based on the library Flint [Har10], that is written in the C programming language. The code is available as a Julia package at `https://github.com/erou/LatticeGFH90.jl`. The package implements Algorithms 12 and 13. The high level manipulations are done directly in Julia while the critical routines are performed by the C library `libembed`. The `libembed` library is part of the Julia package `LatticeGFH90`, it is based on Flint and is compiled against Nemo's version of Flint when `LatticeGFH90` is built. All the tests in this sections were performed on an Intel Core i7-7500U CPU clocked at 2.70GHz, using Nemo 0.19.0 running on Julia 1.5.3, and Nemo's corresponding version of Flint. All the figures were created using gnuplot version 5.2 patchlevel 8. The benchmark functions are available in the file `benchmarks.jl` of the library `LatticeGFH90`, while the data and the gnuplot files are available in the benchmark directory at `https://github.com/erou/thesis/`.

We first measured the time needed to compute Kummer algebras and solutions of (H90), *i.e.* the time needed to perform Algorithm 12, with various small prime p, using the Conway polynomials available in Nemo. It seems that the behaviour of Algorithm 12 is essentially the same, no matter what prime we use, as shown in Figure 7.1 ($p = 3$) and in Figure 7.2 ($p = 11$). Consequently, we only show the experiments made with $p = 3$ in the rest of this section. Note that in Figures 7.1 and 7.2, we compute solutions for (H90) with $m \leq 1000$, but not all the degrees m up to 1000 are computed: indeed we need that $p \nmid m$, and if the level of A_m is a, we

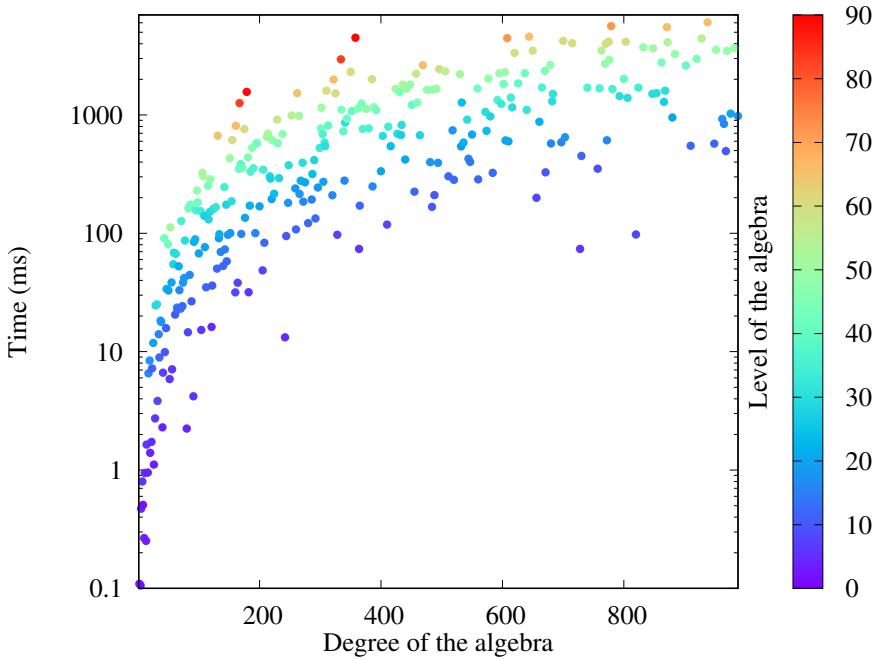


Figure 7.1: Timings for computing decorated Kummer algebras (A_m, α_m) (logarithmic scale) with $p = 3$.

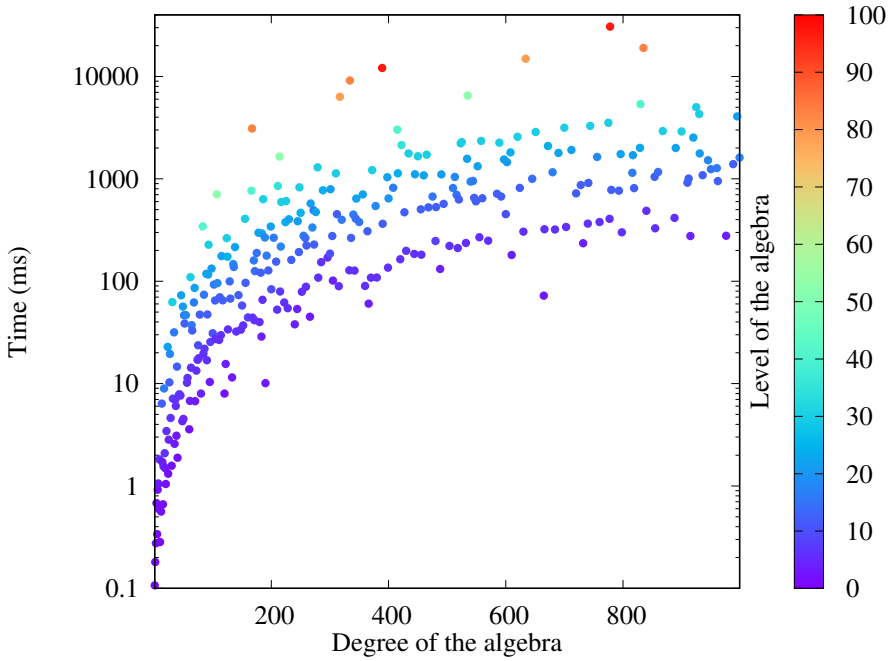


Figure 7.2: Timings for computing decorated Kummer algebras (A_m, α_m) (logarithmic scale) with $p = 11$.

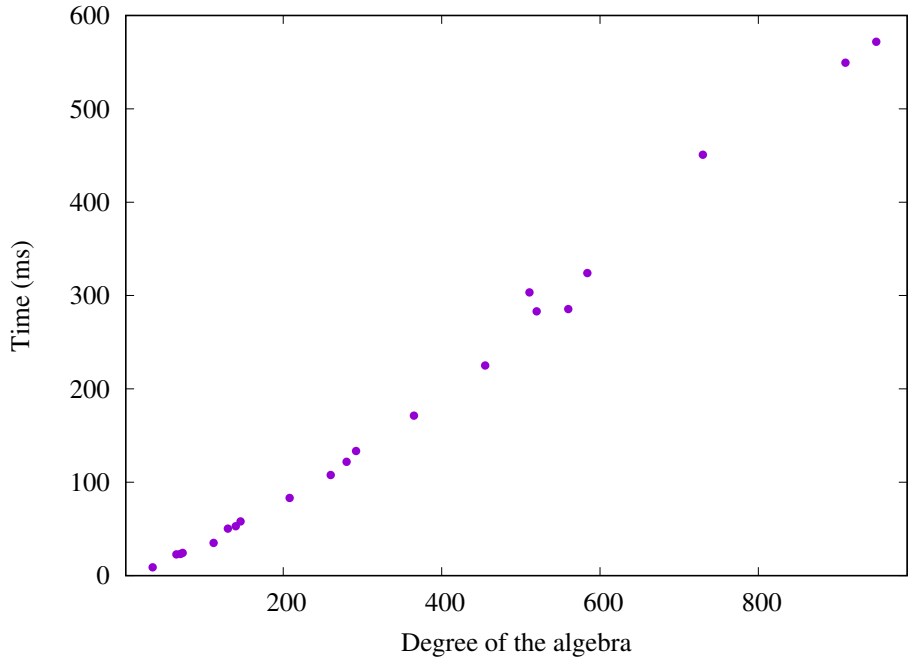


Figure 7.3: Timings for computing decorated Kummer algebras (A_m, α_m) with $p = 3$ and with a given level $a = 12$.

also need the a-th Conway polynomial to be available, which is not always the case. In the complexity analysis of Proposition 7.4.1, the level of the algebra A_m was bounded by its degree m, leading to a quasi-quadratic complexity. However, we see in the timings that this bound is not always relevant in practice. Indeed, the time results show that the level of the algebra has a great impact on the timings: when the level is low, the computations are much faster. In Figure 7.3, we selected only the computations that occured on a fixed level $a = 12$, and we see that in that case the time complexity seems to be quasi-linear in the degree m. We also see that the characteristic p has a small impact on the timings, as shown in Figure 7.4 where we measure the timings for computing the algebra A_{16} and the corresponding solution of (H90) α_{16}, *i.e.* we work with the fixed degree $m = 16$ and we let the characteristic be a prime number p that grows from 3 to 10^4. The bottleneck of Algorithm 12 appears to be the computation of the m-th root extraction routine. When the decoration of two Kummer algebras A_m and A_n, with $m \mid n$, has been done; *i.e.* when Algorithm 12 has been performed and solutions α_m, α_n are available, then the computation of the standard embedding

$$\mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$$

is quite fast. In other words, Algorithm 13 is much faster than Algorithm 12, which is a good thing because we only have to call Algorithm 12 once for each degree m, while Algorithm 13 is called for each embedding computation, and thus can be called several times with the same degree m. In Figure 7.5, we show the timings needed to compute embeddings from $\mathbb{F}_{p^2}$ to $\mathbb{F}_{p^m}$, in the case where $p = 3$, and for every $4 \leq m \leq 1000$ such that $p \nmid m$, $2 \mid m$, and a suitable Conway polynomial is available. Once again, the level of the destination algebra has an important impact on the timings. We also measured the time needed to compute embeddings with extensions of a

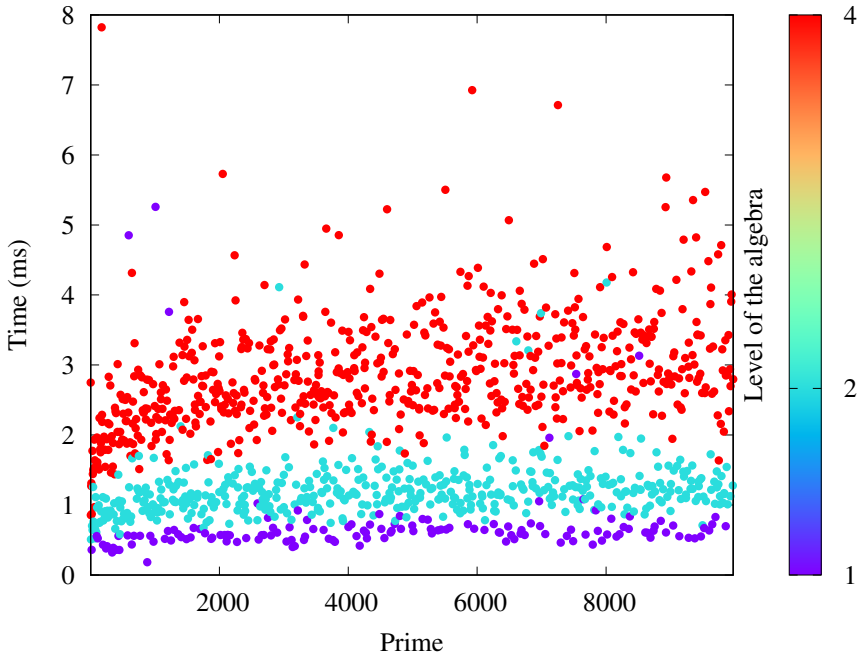


Figure 7.4: Timings for computing decorated Kummer algebras (A_{16}, α_{16}) in characteristic p, with p a prime number satisfying $3 \leq p \leq 10^4$.

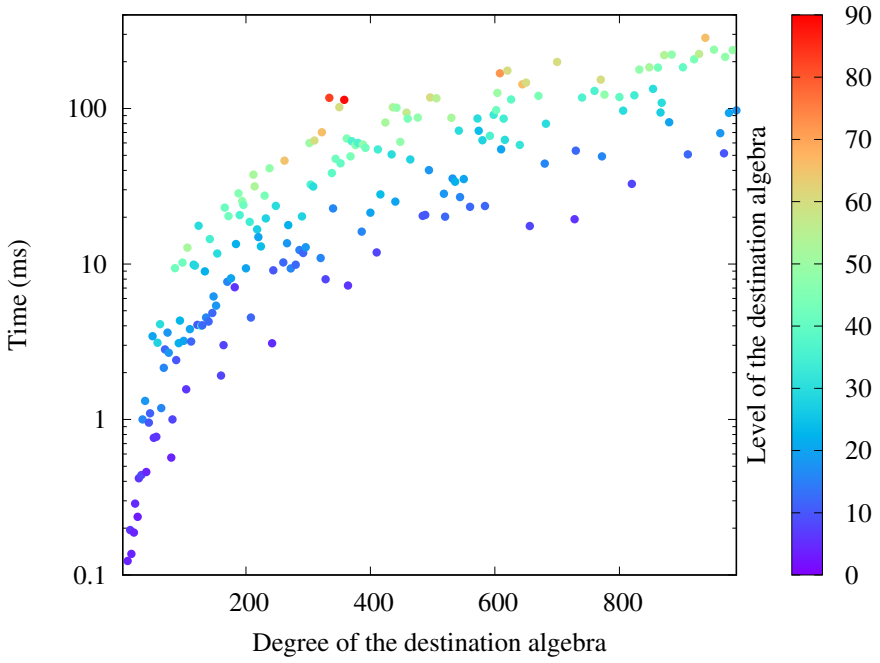


Figure 7.5: Timings to compute the standard embedding from $\mathbb{F}_{p^2}$ to $\mathbb{F}_{p^m}$ (logarithmic scale), for $p = 3$.

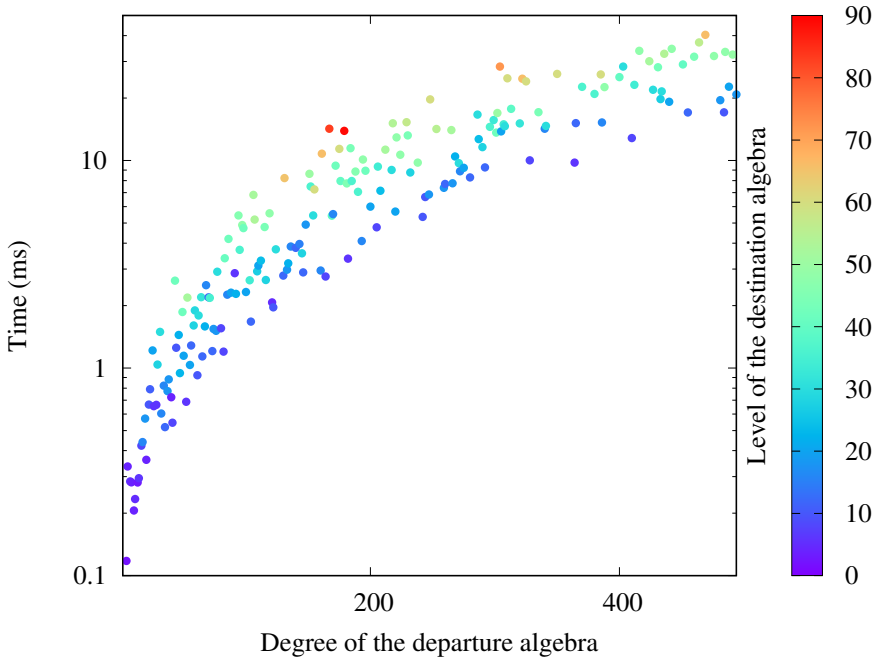


Figure 7.6: Timings to compute the standard embedding from $\mathbb{F}_{p^m}$ to $\mathbb{F}_{p^{2m}}$ (logarithmic scale), for $p = 3$.

fixed degree

$$\left[\mathbb{F}_{p^{2m}} : \mathbb{F}_{p^m}\right] = 2$$

in Figure 7.6, and we obtain similar results as in in Figure 7.5 where the degree of the extension was varying but the base field was fixed. Thus, it appears that the most important parameter is the degree of the destination algebra. Nevertheless, embedding computations seems to be faster with a small extension degree. This is in particular shown in Figure 7.7, where we plot the timings for computing standard embeddings

$$\mathbb{F}_{p^d} \hookrightarrow \mathbb{F}_{p^{880}}$$

with $d \mid 880$. The number 880 was chosen because it is not divisible by $p = 3$ and because it has 20 different divisors, which is the maximum we can obtain for numbers coprime to 3 and less than 1000. The number 560 is also suitable and produces similar results. We use logarithmic scale on the x-axis because there is a greater number of small divisors. The bottleneck of Algorithm 13 for computing a standard embedding

$$\mathbb{F}_{p^m} \hookrightarrow \mathbb{F}_{p^n}$$

seems to be the powering $(\alpha_n)^{\frac{n}{m}}$ occuring in the destination algebra A_n, which explains both the fact that the algorithm is faster when the extension degree $\frac{n}{m}$ is smaller, and the fact that the level of the destination algebra has an important impact of the timings. Again, we see in Figure 7.8 that the impact of the characteristic p on the timings is minimal, compared to the other parameters.

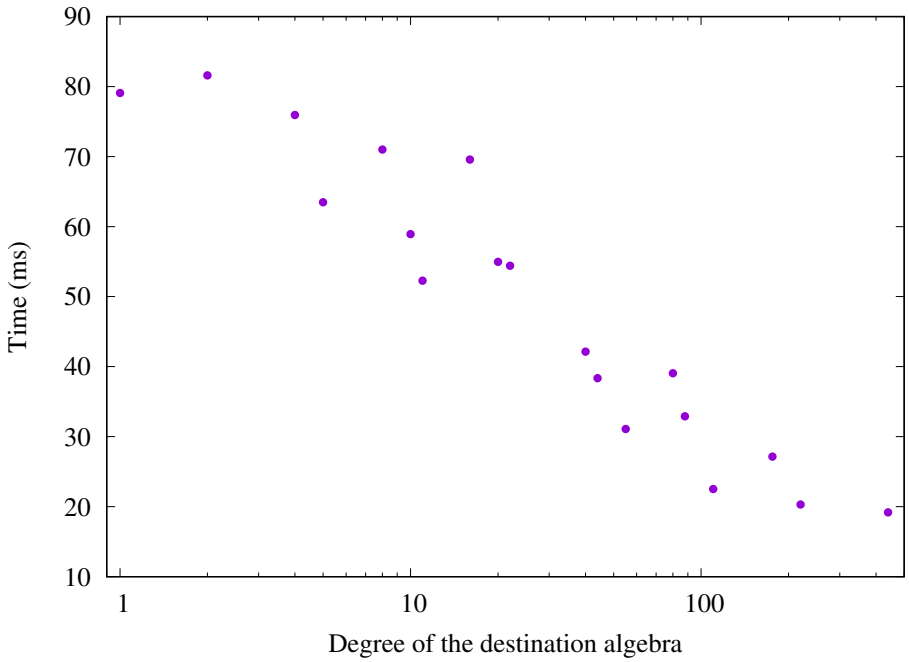


Figure 7.7: Timings to compute the standard embedding from $\mathbb{F}_{p^d}$ to $\mathbb{F}_{p^{880}}$, for $p = 3$ and $d \mid 880$.

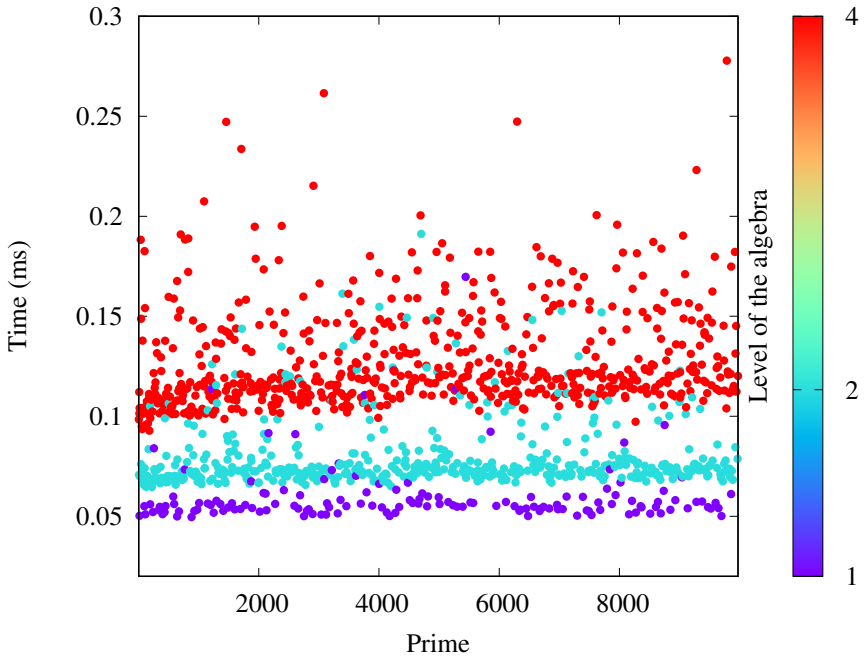


Figure 7.8: Timings to compute the standard embedding from $\mathbb{F}_{p^2}$ to $\mathbb{F}_{p^{16}}$, for p a prime number satisfying $3 \leq p \leq 10^4$.

Conclusion

In this thesis, we investigated the arithmetic of finite field extension from two different angles, the arithmetic of a single finite field extension, and the arithmetic of a lattice of compatibly embedded finite fields.

We studied an alternative, more rigid, kind of bilinear complexity, called hypersymmetric bilinear complexity. Our algorithm to find trisymmetric formulas is very useful to understand and exhibit formulas, but its complexity is prohibitive. We were able to replicate the last known records in the dimension k in which formulas were found, but we were not able to go beyond that point. With more optimization, one could hope to push a little further the computations, but it is likely that new methods have to be found to really make a breakthrough. Asymptocically, we proved that the hypersymmetric complexity of the multiplication in

$$\mathbb{F}_{p^k}$$

is linear in the degree k of the extension, just like with bilinear complexity. However, our approach is clearly not optimal, since we obtain the result as a corollary from the same result in higher dimension. One could hope to obtain better bounds on the hypersymmetric complexity by using an *ad hoc* proof.

We implemented the Bosma-Canon-Steel framework in Nemo, and we introduced a new idea to produce a lattice of compatibly embedded finite fields, based on both Conway polynomials and the Bosma-Canon-Steel framework. Conway polynomials are used in many computer algebra systems, and the Bosma-Canon-Steel framework is used in Magma (and Nemo). Our idea exploits new techniques and is thus interesting in itself. Furthermore, it also leads to a new family of (standard) polynomials that can be used to define finite fields. Nevertheless, this new family has no practical impact at the moment. Indeed, we can prove that computing these polynomials is essentially equivalent to the computation of Conway polynomials. Indeed, with our polynomials, one can recover a standard solution α_m of (H90) and deduce the value

$$(\zeta_{p^a-1})^a.$$

Then, by taking an a-th root, which is done in polynomial time in m, one can find ζ_{p^a-1} and recover the associated Conway polynomial by computing a minimal polynomial. This means that an efficient algorithm to compute our standard polynomials would lead to an efficient algorithm to compute Conway polynomials, which would be unexpected. We thus do not have great hope of finding such an algorithm, however the implemention presented in Section 7.4 is not the only possible way to exploit our definitions. Indeed, it could be possible to loosen the assumption that a cyclotomic lattice is pre-computed, and thus find a middle ground between the rigidity of Conway polynomials and the flexibility of the Bosma-Canon-Steel framework, for example by lazily computing the roots of unity only when needed. An orthogonal line of work would be to construct a complete lattice of compatibly embedded finite fields, *i.e.* working even with degrees that are not coprime with the characteristic p of the base field $\mathbf{k}$.

Bibliography

[All02a] Bill Allombert. Explicit computation of isomorphisms between finite fields. *Finite Fields and Their Applications*, 8(3):332–342, 2002.

[All02b] Bill Allombert. Explicit computation of isomorphisms between finite fields. Revised version. https://www.math.u-bordeaux.fr/~ballombe/fpisom.ps, 2002.

[AW21] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

[Bal98] Stéphane Ballet. *Etude de la complexité bilinéaire de la multiplication dans les corps finis par interpolation sur des courbes algébriques*. PhD thesis, Aix-Marseille 2, 1998.

[Bal99] Stéphane Ballet. Curves with many points and multiplication complexity in any extension of $\mathbb{F}_q$. *Finite Fields and their Applications*, 5:364–377, 1999.

[BCG+17] Alin Bostan, Frédéric Chyzak, Marc Giusti, Romain Lebreton, Grégoire Lecerf, Bruno Salvy, and Éric Schost. *Algorithmes efficaces en calcul formel*. Published by the authors, 2017.

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[BCS97] Wieb Bosma, John Cannon, and Allan Steel. Lattices of compatibly embedded finite fields. *Journal of Symbolic Computation*, 24(3-4):351–369, 1997.

[BCS13] Peter Bürgisser, Michael Clausen, and Mohammad A Shokrollahi. *Algebraic complexity theory*, volume 315. Springer Science & Business Media, 2013.

[BDEZ12] Razvan Barbulescu, Jérémie Detrey, Nicolas Estibals, and Paul Zimmermann. Finding optimal formulae for bilinear maps. In *International Workshop on the Arithmetic of Finite Fields*, pages 168–186. Springer, 2012.

[BDFD+17] Ludovic Brieulle, Luca De Feo, Javad Doliskani, Jean-Pierre Flori, and Éric Schost. Computing isomorphisms and embeddings of finite fields. *arXiv preprint arXiv:1705.01221*, 2017.

[BFSS06] Alin Bostan, Philippe Flajolet, Bruno Salvy, and Éric Schost. Fast computation of special resultants. *Journal of Symbolic Computation*, 41(1):1–29, 2006.

[BK78] Richard P Brent and Hsiang T Kung. Fast algorithms for manipulating formal power series. *Journal of the ACM (JACM)*, 25(4):581–595, 1978.

[BLS03] Alin Bostan, Grégoire Lecerf, and Éric Schost. Tellegen's principle into practice. In *Proceedings of the 2003 international symposium on Symbolic and algebraic computation*, pages 37–44, 2003.

[BPR+21] Stéphane Ballet, Julia Pieltant, Matthieu Rambaud, Hugues Randriambololona, Robert Rolland, and Jean Chaumine. On the tensor rank of multiplication in finite extensions of finite fields and related issues in algebraic geometry. *Russian Mathematical Surveys*, 76(1):29, 2021.

[BR04] Stéphane Ballet and Robert Rolland. Multiplication algorithm in a finite field and tensor rank of the multiplication. *Journal of Algebra*, 272:173–185, 2004.

[Bsh13] Nader H. Bshouty. Multilinear complexity is equivalent to optimal tester size. *Electronic Colloquium on Computational Complexity*, 20:11, 2013.

[CC88] David V. Chudnovsky and Gregory V. Chudnovsky. Algebraic complexities and algebraic curves over finite fields. *Journal of Complexity*, 4(4):285–316, 1988.

[CK91] David G Cantor and Erich Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28(7):693–701, 1991.

[CL13] Jean-Marc Couveignes and Reynald Lercier. Fast construction of irreducible polynomials over finite fields. *Israel Journal of Mathematics*, 194(1):77–105, 2013.

[CÖ10] Murat Cenk and Ferruh Özbudak. On multiplication in finite fields. *Journal of Complexity*, 26(2):172–186, 2010.

[Cov19] Svyatoslav Covanov. Improved method for finding optimal formulas for bilinear maps in a finite field. *Theoretical Computer Science*, 2019.

[CT65] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[CW90] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation*, 9:251–280, 1990.

[Dev21] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.2)*, 2021. `http://www.sagemath.org`.

[DF10] Luca De Feo. *Algorithmes rapides pour les tours de corps finis et les isogenies*. PhD thesis, PhD thesis. Ecole Polytechnique X, 2010.

[DFDS13] Luca De Feo, Javad Doliskani, and Éric Schost. Fast algorithms for l-adic towers over finite fields. In *Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation*, pages 165–172, 2013.

[DFDS14] Luca De Feo, Javad Doliskani, and Éric Schost. Fast arithmetic for the algebraic closure of finite fields. In *ISSAC '14*, pages 122–129. ACM, 2014.

[DFRR18] Luca De Feo, Hugues Randriambololona, and Édouard Rousseau. Lattices of compatibly embedded finite fields in nemo/flint. *ACM Communications in Computer Algebra*, 52(2):38–41, 2018.

[DFRR19] Luca De Feo, Hugues Randriam, and Édouard Rousseau. Standard lattices of compatibly embedded finite fields. In *Proceedings of the 2019 on International Symposium on Symbolic and Algebraic Computation*, pages 122–130, 2019.

[DFS10] Luca De Feo and Eric Schost. transalpyne: a language for automatic transposition. *ACM Communications in Computer Algebra*, 44(1/2):59–71, 2010.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, Nov 1976.

[DR99] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.

[DR02] Joan Daemen and Vincent Rijmen. *The design of Rijndael*, volume 2. Springer, 2002.

[Gao93] Shuhong Gao. *Normal Bases over Finite Fields*. PhD thesis, University of Waterloo, 1993.

[GKZ14] Robert Granger, Thorsten Kleinjung, and Jens Zumbrägel. On the powers of 2. *IACR Cryptology ePrint Archive*, 2014:300, 2014.

[Gop81] Valerii Denisovich Goppa. Codes on algebraic curves. In *Soviet Math. Dokl.*, volume 24, pages 170–172, 1981.

[H+16] William Hart et al. Nemo Package (Version 0.5.0). http://nemocas.org, 2016.

[Har09] David Harvey. Faster polynomial multiplication via multipoint kronecker substitution. *Journal of Symbolic Computation*, 44(10):1502–1510, 2009.

[Har10] William Hart. Fast library for number theory: an introduction. *Mathematical Software-ICMS 2010*, pages 88–91, 2010.

[HL04] Lenwood S Heath and Nicholas A Loehr. New algorithms for generating Conway polynomials over finite fields. *Journal of Symbolic Computation*, 38(2):1003–1024, 2004.

[HVDH19a] David Harvey and Joris Van Der Hoeven. Integer multiplication in time O (n log n). 2019.

[HVDH19b] David Harvey and Joris Van Der Hoeven. Polynomial multiplication over finite fields in time O (n log n). 2019.

[Jul] Julia : a high-level, high-performance dynamic language for technical computing. http://julialang.org.

[Kal87] Erich Kaltofen. Computer algebra algorithms. *Annual review of computer science*, 2(1):91–118, 1987.

[Kar63] Anatolii Karatsuba. Multiplication of multidigit numbers on automata. In *Soviet Physics Doklady*, volume 7, pages 595–596, 1963.

[KS97] Erich Kaltofen and Victor Shoup. Fast polynomial factorization over high algebraic extensions of finite fields. In *Proceedings of the 1997 international symposium on Symbolic and algebraic computation*, pages 184–188, 1997.

[KU08] Kiran S Kedlaya and Christopher Umans. Fast modular composition in any characteristic. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 146–155. IEEE, 2008.

[KU11] Kiran S Kedlaya and Christopher Umans. Fast polynomial factorization and modular composition. *SIAM Journal on Computing*, 40(6):1767–1802, 2011.

[Lan04] Serge Lang. *Algebra*, volume 211. Springer, 2004.

[LJ91] Hendrik W Lenstra Jr. Finding isomorphisms between finite fields. *Mathematics of Computation*, pages 329–347, 1991.

[LN97] Rudolf Lidl and Harald Niederreiter. *Finite fields*, volume 20. Cambridge university press, 1997.

[Moe76] Robert T Moenck. Another polynomial homomorphism. *Acta Informatica*, 6(2):153–169, 1976.

[MVOV18] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 2018.

[Pap03] Christos H Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[Par] Richard Parker. Finite fields and Conway polynomials. 1990. *Vortrag am IBM Scientific Center Heidelberg*.

[Per96] Daniel Perrin. *Cours d'algèbre*, volume 30. Ellipses Paris, 1996.

[Pie12] Julia Pieltant. Tours de corps de fonctions algébriques et rang de tenseur de la multiplication dans les corps finis. *PhD of Université d'Aix-Marseille, Institut de Mathématiques de Luminy*, 2012.

[Pin92] RICHARD GE Pinch. Recognising elements of finite fields. In *Cryptography and coding, II*, pages 193–197. Citeseer, 1992.

[PS06] Cyril Pascal and Éric Schost. Change of order for bivariate triangular sets. In *Proceedings of the 2006 international symposium on Symbolic and algebraic computation*, pages 277–284, 2006.

[PS13] Adrien Poteaux and Éric Schost. Modular composition modulo triangular sets and applications. *computational complexity*, 22(3):463–516, 2013.

[Rai96] Eric M. Rains. Efficient computation of isomorphisms between finite fields. personal communication, 1996.

[Ran12] Hugues Randriambololona. Bilinear complexity of algebras and the chudnovsky-chudnovsky interpolation method. *Journal of Complexity*, 28(4):489–517, 2012.

[Ran15] Hugues Randriambololona. On products and powers of linear codes under componentwise multiplication. In *Algorithmic Arithmetic, Geometry, and Coding Theory*, volume 637 of *Contemporary Mathematics*, pages 3–78. AMS, 2015.

[RR21] Hugues Randriambololona and Édouard Rousseau. Trisymmetric multiplication formulae in finite fields. In Jean Claude Bajard and Alev Topuzoğlu, editors, *Arithmetic of Finite Fields*, pages 92–111, Cham, 2021. Springer International Publishing.

[RS60] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[RSA78] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[Sch92] Alfred Scheerhorn. Trace-and norm-compatible extensions of finite fields. *Applicable Algebra in Engineering, Communication and Computing*, 3(3):199–209, 1992.

[Sha48] Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[Sho94] Victor Shoup. Fast construction of irreducible polynomials over finite fields. *Journal of Symbolic Computation*, 17(5):371–391, 1994.

[Sho95] Victor Shoup. A new polynomial factorization algorithm and its implementation. *J. Symb. Comput.*, 20(4):363–397, 1995.

[Sho99] Victor Shoup. Efficient computation of minimal polynomials in algebraic extensions of finite fields. In *Proceedings of the 1999 international symposium on Symbolic and algebraic computation*, pages 53–58, 1999.

[SL84] Gadiel Seroussi and Abraham Lempel. On symmetric algorithms for bilinear forms over finite fields. *Journal of Algorithms*, 5:327–344, 1984.

[SS71] Arnold Schönhage and Volker Strassen. Fast multiplication of large numbers. *Computing*, 7:281–292, 1971.

[Sti09] Henning Stichtenoth. *Algebraic function fields and codes*, volume 254. Springer Science & Business Media, 2009.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[STV92] Igor E. Shparlinski, Michael A. Tsfasman, and Serge G. Vladut. Curves with many points and multiplication in finite fileds. In *Coding Theory and Algebraic Geometry*, volume 1518 of *Lecture Notes in Mathematics*, pages 145–169. Springer, 1992.

[VZGG13] Joachim Von Zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge university press, 2013.

[VZGS92] Joachim Von Zur Gathen and Victor Shoup. Computing frobenius maps and factoring polynomials. *Computational complexity*, 2(3):187–224, 1992.

[Win71] Shmuel Winograd. On multiplication of 2×2 matrices. *Linear Algebra and its Applications*, 4:381–388, 1971.

[Win77] Shmuel Winograd. Some bilinear forms whose multiplicative complexity depends on the field of constants. *Mathematical Systems Theory*, 10:169–180, 1977.

ECOLE
DOCTORALE
DE MATHEMATIQUES
HADAMARD

Titre : Arithmétique efficace des extensions de corps finis

Mots clés : Calcul formel, corps finis, extensions

Résumé : Les corps finis sont omniprésents en cryptographie et en théorie des codes, deux domaines de première importance dans les communications modernes. Ainsi, il est crucial de représenter les corps finis et d'y faire des calculs de la façon la plus efficace possible. Dans cette thèse, nous travaillons sur l'arithmétique des extensions de corps finis, de deux manières différentes et indépendantes.
Dans la première partie, nous étudions l'arithmétique d'une unique extension de corps fini $\mathbb{F}_{p^k}$. Une manière d'estimer la complexité d'un algorithme dans une extension est de compter le nombre de multiplications effectuées dans le corps de base $\mathbb{F}_p$. Ce modèle est connu sous le nom de complexité bilinéaire. Dans cette thèse, nous généralisons les résultats connus pour la complexité bilinéaire à de nouveaux types de complexités, qualifiées de complexité hypersymétrique et de complexité multilinéaire. Nous fournissons un algorithme pour calculer la complexité hypersymétrique de la multiplication dans l'extension $\mathbb{F}_{p^k}$, ainsi qu'une implémentation et son analyse.
Dans la seconde partie, notre but est de construire une structure de donnée efficace pour représenter plusieurs extensions simultanément, ainsi que les plongements entre elles. Ces plongements sont rarement uniques, et il faut donc s'assurer que les choix effectués soient compatibles, c'est-à-dire que dès lors que trois corps finis sont en jeu, le plongement calculé est indépendant du chemin parcouru. Nous donnons une implémentation de l'algorithme de Bosma-Canon-Steel, qui permet d'obtenir des plongements compatibles, et qui était uniquement disponible dans MAGMA, mais est désormais disponible dans le système de calcul formel Nemo. Nous proposons également une nouvelle construction nommée réseau standard de corps finis compatiblement plongés, inspirée de l'algorithme de Bosma-Canon-Steel et des polynômes de Conway, qui sont à la base d'une autre méthode populaire pour obtenir la compatibilité. Contrairement à l'utilisation des polynômes de Conway, notre méthode permet d'utiliser des corps finis définis de manière arbitraire, tout en restant efficace. Nous analysons en détails la complexité de nos algorithmes, et donnons une implémentation montrant que notre construction peut être utilisée en pratique.

Title : Efficient Arithmetic of Finite Field Extensions

Keywords : Computer algebra, finite fields, extensions

Abstract : Finite fields are ubiquitous in cryptography and coding theory, two fields that are of utmost importance in modern communications. For that reason, it is crucial to represent finite fields and compute in them in the most efficient way possible. In this thesis, we investigate the arithmetic of finite field extensions in two different and independent ways.
In the first part, we study the arithmetic of one fixed finite field extension $\mathbb{F}_{p^k}$. A way of estimating the complexity of an algorithm in an extension is to count the number of multiplications performed in the base field $\mathbb{F}_p$. This model is called bilinear complexity. In this thesis, we generalize results on bilinear complexity to other types of complexities called hypersymmetric complexity and multilinear complexity. We give an algorithm to compute the hypersymmetric complexity in the extension $\mathbb{F}_{p^k}$, as well as an implementation and its analysis.
In the second part, out goal is to construct an efficient data structure to represent several extensions simultaneously, as well as embeddings between them. These embeddings are rarely unique, thus we have to make sure that our choices are compatible: i.e. each time three extensions are involved the embedding that is computed does not depend on the actual path taken in the lattice of extensions. We give an implementation of Bosma, Canon and Steel's algorithm, which produces compatible embeddings, and which was originally only available in MAGMA, but is now in the free computer algebra software Nemo. We also propose a new construction called standard lattice of compatibly embedded finite fields, inspired by both Bosma, Canon and Steel's algorithm and Conway polynomials. Another popular method to obtain compatibility is based on these polynomials, but contrary to Conway polynomials, our method enables to work with user-defined finite fields, while being efficient. We give an analysis of our algorithms, as well as an implementation that shows that our construction is practical.

Institut Polytechnique de Paris
91120 Palaiseau, France