Lattices of compatibly embedded finite fields in Nemo/Flint

Luca De Feo[*], Hugues Randriambololona[†], Édouard Rousseau[‡]

Abstract

We present a software that allows the user to compute embeddings between arbitrary finite fields. It is written using Julia and C, as part of Nemo and Flint[7, 6]. The software ensures compatibility between the embeddings, so that the user can work with multiple fields and have coherent results independently of where the computations are made and of where the results are expressed.

Finite fields are widely used in mathematics and their applications. They are often the building block of more complicated algebraic structure used in cryptology or coding theory. As a consequence, several computer algebra systems or libraries have been written in order to work in arbitrary finite fields. Among them, we find Magma [1], Sage [5], Flint [6], NTL [10] or PARI [11]. Many problems require not only to work in a given finite field K, but also in finite extensions of K, as represented in Figure 1, that are again finite fields, or in the algebraic closure of K. In particular, it is desirable to be able to freely *move* from a field to a subfield or an extension.

Given two finite fields E and F with cardinalities $\#E = p^m$ and $\#F = p^n$, we know that E can be embedded in F if and only if $m \mid n$. In other words, E is in that case isomorphic to a subfield $E' \subset F$ of F with cardinality $\#E' = p^m$. There are $m = [E : \mathbb{F}_p] = \# \operatorname{Gal}(E/\mathbb{F}_p)$ distinct embeddings from E to F (the degree of E over $\mathbb{F}_p$ will also be denoted by $\partial(E)$). Indeed, the Galois group of the extension E over $\mathbb{F}_p$ acts on the embeddings and, given two different embeddings $\phi_{E\hookrightarrow F}$ and $\phi'_{E\hookrightarrow F}$ and an element $x \in E$, the images $\phi_{E\hookrightarrow F}(x)$ and $\phi'_{E\hookrightarrow F}(x)$ must be conjugate. There is no canonical embedding from E to F. Furthermore, the proof of the fact that E can be embedded in F if and only if $\partial(E) \mid \partial(F)$ does not give an efficient algorithm. Finding such an algorithm is an interesting problem, and there exists a variety of solutions, see for example the survey paper [3].

Additionaly, we want the embeddings to be compatible. Given three finite fields E, F, and G, such that $\partial(E) \mid \partial(F)$ and $\partial(F) \mid \partial(G)$, and three embeddings $\phi_{E\hookrightarrow F}$, $\phi_{F\hookrightarrow G}$, and $\phi_{E\hookrightarrow G}$, we say that the embeddings are *compatible* if

$$\phi_{E\hookrightarrow G} = \phi_{F\hookrightarrow G} \circ \phi_{E\hookrightarrow F}.$$

In other words, we want the diagram of Figure 2 to commute. In the context of a computer algebra system, this condition is important in order to give the user coherent answers when performing operations in different fields. This is the case when computing in the algebraic closure of a finite field, because the ambient field may change often. There are also applications in isogeny-based [4] or pairing-based cryptography. We note $E \hookrightarrow F$ if E is explicitly embedded in F, *i.e.* if we have computed an embedding $\phi_{E\hookrightarrow F}$.

[*]Université de Versailles and Inria, Paris Saclay, `luca.de-feo@uvsq.fr`

[†]Télécom ParisTech, `randriam@telecom-paristech.fr`

[‡]Télécom ParisTech and Université de Versailles, `edouard.rousseau@telecom-paristech.fr`

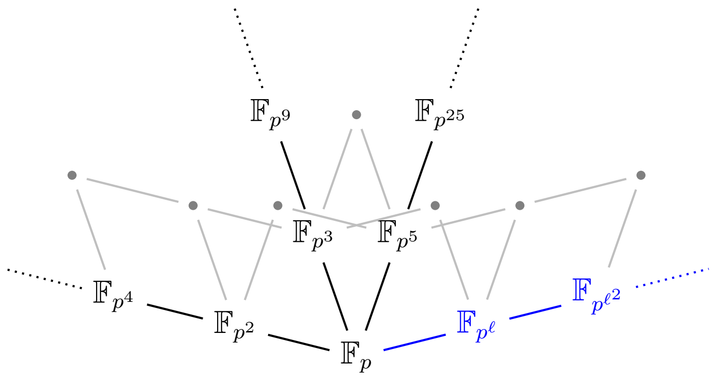

Figure 1: Extensions of $\mathbb{F}_p$.

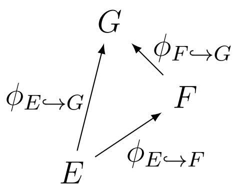
$$\begin{tikzcd} & & G \\ & & & F \arrow[ul, "\phi_{F\hookrightarrow G}"'] \\ & E \arrow[uur, "\phi_{E\hookrightarrow G}"] \arrow[ur, "\phi_{E\hookrightarrow F}"'] \end{tikzcd}$$

Figure 2: Embeddings between finite fields.

Compatibility Compatibility can be achieved using *Conway polynomials* [9, 8], a set $\{P_d\}_{d\in D}$ of irreducible polynomials over $\mathbb{F}_p$ such that any root β_d of P_d is primitive in $\mathbb{F}_p[X]/(P_d(X)) \cong \mathbb{F}_{p^d}$ and such that if $d_1, d_2 \in D$ and $d_1|d_2$, then $N_{\mathbb{F}_{p^{d_2}}/\mathbb{F}_{p^{d_1}}}(\beta_{d_2})$ is a root of P_{d_1}, where $N_{\mathbb{F}_{p^{d_2}}/\mathbb{F}_{p^{d_1}}}$ denotes the norm of $\mathbb{F}_{p^{d_2}}$ over $\mathbb{F}_{p^{d_1}}$. A compatible embedding of $\mathbb{F}_{p^{d_1}}$ in $\mathbb{F}_{p^{d_2}}$ is then obtained by sending β_{d_1} to $\beta_{d_2}^k$, with $k = (p^{d_2} - 1)/(p^{d_1} - 1)$. By adding a minimality condition with respect to some ordering, this provides a way of constructing standardized finite field extensions. However Conway polynomials are hard to compute, so in practice this technique can only be used with rather small fields.

Bosma, Cannon and Steel suggested another scheme [2] in which they give an axiomatic characterization of a lattice of compatibly embedded finite fields. Their method, implemented in Magma, does not require any precomputation and enables one to work in any user-defined finite field. However, embedding a new finite field requires polynomial factorization and leads to a cost strictly larger than quadratic in the extension degree [3]. Following [2], we say that a pair $\mathfrak{L} = (L, \Phi)$, where L is a set of finite fields and Φ is a set of embeddings between elements of L, is a *lattice of compatibly embedded finite fields* if

CE1 (unicity) for each pair (E, F) of elements in L, there exists at most one corresponding embedding $\phi_{E\hookrightarrow F} \in \Phi$.

CE2 (reflexivity) For each $E \in L$, the identity map $\mathrm{Id}_E = \phi_{E\hookrightarrow E}$ is in Φ.

CE3 (prime subfield) There is exactly one $P \in L$ such that $\partial(P) = 1$, and for all $F \in L$, there exists $\phi_{P\hookrightarrow F} \in \Phi$

CE4 (invertibility) If $E \hookrightarrow F$ and $\partial(E) = \partial(F)$, then $F \hookrightarrow E$ and $\phi_{F\hookrightarrow E} = \phi^{-1}_{E\hookrightarrow F}$.

CE5 (transitivity) For any triple (E, F, G) of elements in L, if $E \hookrightarrow F \hookrightarrow G$ then $E \hookrightarrow G$ and $\phi_{E\hookrightarrow G} = \phi_{F\hookrightarrow G} \circ \phi_{E\hookrightarrow F}$.

CE6 (intersections) For each $E, F, G \in L$ such that $F \hookrightarrow G$ and $E \hookrightarrow G$, there exists $S \in L$ such that $\partial(S) = \gcd(\partial(E), \partial(F))$ and $S \hookrightarrow E$, $S \hookrightarrow F$.

Under those conditions, we can prove [2] that we are able to add a finite field in L or an embedding that is not yet in Φ without altering the compatibility of the lattice $\mathfrak{L}$.

Our contribution We implemented Bosma, Canon and Steel framework using Nemo/Flint [7, 6] and following conditions CE1 to CE6. These conditions are, for most of them, very natural. The condition CE3 is technical and does not imply any work in our implementation because finite fields elements in Nemo/Flint are represented by polynomials over $\mathbb{F}_p$, so the embedding of $\mathbb{F}_p$ into an extension is trivial. Finally, condition CE6 ensures that the implicit isomorphisms between subfields are made explicit. In order to meet this last condition, when embedding a finite field F in G, for each subfield S of G, we check that the finite field $S \cap F$ is embedded in S and F, and if not, we embed it. If there is not any finite field of degree $d = \gcd(\partial(S), \partial(F))$, we compute an arbitrary finite field I of degree d using Flint and we embed I in S and F, resulting in a recursive call to our embedding algorithm.

Our code is available as an experimental branch of Nemo[1], that is itself based on an experimental branch of Flint[2]. Critical routines (*e.g.* polynomial factorization, matrix computations) are written in C and computed by Flint, whereas high level tasks (*e.g.* checking conditions CE5 and CE6) are written in Julia using Nemo. Our goal is to first include the C code inside the standard Flint library and then the Julia code in Nemo. With our experimental code, it is possible to define arbitrary finite fields and to compute compatible embeddings between them, it is also possible to compute a section of a field to a subfield: *i.e.* a map that sends an element to its inverse image when the element is in the subfield, and throw an error otherwise. All these computations are automatic and tranparent to the user, except if he or she wants to work with the maps themselves. All the computed embeddings are kept in memory so that the same work is not done twice. Together with conditions CE5 and CE6, this leads to many embeddings being stored behind the scenes. Here is an example of a minimal session using our code[3].

```
In:  p = 5
     # We create k2 = GF(25) = GF(5)[x2]
     k2, x2 = FiniteField(p, 2, ‘‘x2’’)
     k4, x4 = FiniteField(p, 4, ‘‘x4’’)
     k8, x8 = FiniteField(p, 8, ‘‘x8’’)
     # We compute the embedding k2->k4
     f2_4 = embed(k2, k4)
     y = f2_4(x2)
Out: x4^3+x4^2+x4+3

In:  # We check that y is in GF(25)
     y^(p^2) ==  y
Out: true

In:  f2_8 = embed(k2, k8)
     f4_8 = embed(k4, k8)
     # We check the compatibility
     f2_8(x2) == f4_8(f2_4(x2))
Out: true

In:  # We can directly see x4^2+1
     # as an element of k8
     z = k8(x4^2+1)
Out: 3*x8^6+4*x8^5+3*x8^3+4*x8+2

In:  k4(z)
Out: x4^2+1
```

Future work The code has yet to be optimised: condition CE5 could be fulfilled lazily by computing the embeddings only when needed by the user. This would prevent the storage of useless embeddings. Moreover, the embedding algorithm used by Bosma, Canon and Steel (the naive algorithm) is not optimal either. Replacing the naive algorithm by a more sophisticated algorithm requires both theoretical work and some new implementations. An important question is whether

[1]`https://github.com/erou/Nemo.jl/tree/embeddings`

[2]`https://github.com/defeo/flint2/tree/embeddings`

[3]A longer and interactive one is available at `https://mybinder.org/v2/gh/defeo/Nemo-embeddings-demo/master?filepath=demo.ipynb`.

a "standardized" construction can be found for these algorithms, similar to the one obtained using Conway polynomials. Contrary to Bosma, Canon and Steel framework, Conway polynomials also permit to simply obtain the generator of a field from the generator of another field using norms. We aim to be able to do the same thing, by memorizing a little more than the generators of our finite fields.

References

[1] 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).

[2] Wieb Bosma, John Cannon, and Allan Steel. Lattices of compatibly embedded finite fields. *Journal of Symbolic Computation*, 24(3-4):351–369, 1997.

[3] 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.

[4] Luca De Feo. Mathematics of isogeny based cryptography. *arXiv preprint arXiv:1711.04062*, 2017.

[5] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.1)*, 2017. `http://www.sagemath.org`.

[6] William Hart. Fast library for number theory: an introduction. *Mathematical Software-ICMS 2010*, pages 88–91, 2010.

[7] William Hart et al. Nemo Package (Version 0.5.0). `http://nemocas.org`, 2016.

[8] Lenwood S Heath and Nicholas A Loehr. New algorithms for generating Conway polynomials over finite fields. 1998.

[9] Richard Parker. Finite fields and Conway polynomials. 1990. *Vortrag am IBM Scientific Center Heidelberg.*

[10] Victor Shoup. NTL: A library for doing number theory. `http://www.shoup.net/ntl`.

[11] The PARI Group, Univ. Bordeaux. *PARI/GP version* `2.9.4`, 2018. available from `http://pari.math.u-bordeaux.fr/`.