

FORMALIZING CARLESON’S THEOREM IN LEAN

LARS BECKER, MARÍA INÉS DE FRUTOS-FERNÁNDEZ, LEO DIEDERING,
FLORIS VAN DOORN, SÉBASTIEN GOUËZEL, EVGENIA KARUNUS, EDWARD VAN DE MEENT,
PIETRO MONTICONE, JASPER MULDER-SOHN, JIM PORTEGIES, JORIS ROOS,
MICHAEL ROTHGANG, JAMES SUNDSTROM, AND JEREMY TAN

ABSTRACT. We present the formalization of Carleson’s theorem in the proof assistant *Lean*. This paper describes the mathematical content, organization of the project, the blueprint, and the main design decisions behind the formalization. It is the result of a large collaborative effort, written and developed in public.

CONTENTS

1. Introduction	2
1.1. Brief history of Carleson’s theorem and significance	2
1.2. Overview of the formalization project	2
1.3. Related work	2
2. The Metric Space Carleson theorem	3
2.1. Mathematical background	3
2.2. Statement of the main results	4
3. Project organization	4
4. Working with a blueprint	4
4.1. The blueprint writing process	4
4.2. Changes and refinements	4
4.3. Constant tweaking	5
4.4. Dealing with mistakes	5
4.5. Lessons learned	5
5. Design decisions	5
5.1. Treatment of constants	5
5.2. Standing assumptions and the <code>ProofData</code> pattern	6
5.3. Working with real numbers	7
5.4. Use of <code>ENorm</code>	8
5.5. Working with L^p functions	9
5.6. Test function classes	9
5.7. Common pitfalls	10
6. Conclusion	10
References	10

1. INTRODUCTION

1.1. Brief history of Carleson’s theorem and significance.

1.2. Overview of the formalization project. The goal of the Carleson project was to formalize the proof of the metric space Carleson theorem [ref], proven in [Bec+24a], as well as the proof of the classical Carleson theorem [ref] as a corollary of this more general result.

The original reference [Bec+24a] was written as a traditional paper for experts in harmonic analysis, but its formalization was since the beginning envisioned as a large scale collaborative project. This posed a problem, since there are not many experts in formalization that are also experts in harmonic analysis, so most of the potential contributors to the formalization would not have the required expertise to fill in the gaps that are considered routine arguments in a harmonic analysis paper.

To address this challenge, before the formalization started, the authors of [Bec+24a] wrote a much more detailed document [Bec+24b] intended as a blueprint for the formalization, which was modified as needed while the formalization process was taking place (see §4).

The Carleson formalization project was publicly announced and opened to contributors in June 2024, and it was completed in July 2025. The core authors of the formalization are the 14 authors of this paper, and other smaller contributions were made by the 14 people listed in the acknowledgements section. This makes the Carleson project one of the largest scale formalization efforts up to date.

The formalization was written in Lean 4, making extensive use of Lean’s mathematical library, `Mathlib`. The code was developed in a public GitHub repository¹, with an associated website² that also linked to HTML and PDF versions of the blueprint. Coordination among project collaborators took place using the Lean community Zulip chat³. For more details, see §3.

The project comprises around 38,000 lines of Lean code (not including spaces or comments), of which around 12,000 are intended to be upstreamed to the `Mathlib` library, a process that is still ongoing. Pull requests into `Mathlib` coming from this project are marked with the `carleson` tag; at the time of writing this article, this includes 90 merged pull requests and 11 more under review.

1.3. Related work.

- Other large-scale formalizations in Lean.
- Other harmonic analysis formalizations.

Acknowledgement. The authors are grateful to Asgar Janneshan, Rajula Srivastava and Christoph Thiele for writing the mathematical paper [cite] and the blueprint [cite] on which this formalization is based. We furthermore acknowledge contributions in the form of small formalization additions, pointing out corrections to the blueprint, or supplying ideas to the Lean efforts by the following people: Michel Alexis, Bolton Bailey, Julian Berman, Joachim Breitner, Martin Dvořák, Georges Gonthier, Aaron Hill, Austin Letson, Bhavik Mehta, Eric Paul, Clara Torres, Dennis Tsar, Andrew Yang, and Ruben van de Velde.

L.B., M.I.d.F.F., L.D., F.v.D., M.R. were funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy – EXC-2047/1

¹<https://github.com/fpvandoorn/carleson>

²<https://florisvandoorn.com/carleson/>

³<https://leanprover.zulipchat.com>

– 390685813. L.B. was also supported by SFB 1060. J.R. was supported in part by NSF grant DMS-2154835 and a HIM fellowship for the Fall 2024 trimester program in Bonn.

2. THE METRIC SPACE CARLESON THEOREM

2.1. Mathematical background. The basic object of study in Carleson's theorem is the Fourier series of a periodic function on the real numbers. Given a function $f : [0, 2\pi) \rightarrow \mathbb{C}$ and $n \in \mathbb{Z}$, we first define the n th Fourier coefficient of f as

$$(1) \quad c_n := \widehat{f}_n := \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-inx} dx.$$

The Lean version `fourierCoeff` uses `Mathlib`'s formalization of the Bochner integral and thus is well-defined if $f \in L^1(0, 2\pi)$.

We define the N -th partial Fourier sum of f at $x \in [0, 2\pi]$ as

$$(2) \quad S_N f(x) := \sum_{n=-N}^N c_n e^{inx}$$

and its Fourier series as $\lim_{N \rightarrow \infty} S_N f(x)$.

The metric space Carleson theorem in [\[Bec+24b\]](#) is a restricted weak type estimate for a generalized Carleson operator. One of the generalizations here is that the domain of the functions operated on can be any *doubling metric measure space* (X, ρ, μ, a) . This is defined to be a complete and locally compact metric space (X, ρ) equipped with a non-zero locally finite Borel measure μ that satisfies the following doubling condition for $a \in \mathbb{N}$. For all $x \in X$ and all $R > 0$ we have

$$(3) \quad \mu(B(x, 2R)) \leq 2^a \mu(B(x, R)),$$

where $B(x, R) := \{y \in X : \rho(x, y) < R\}$ denotes the open ball of radius R centered at x (see `DoublingMeasure`).

The proof of the metric space Carleson theorem relies on several tools from real analysis and part of the formalization is a proof of the Marcinkiewicz real interpolation theorem (see [4.2.4](#)). Statement and proof require the following definitions for which we could rely on `Mathlib`'s measure theory (see e.g. [\[Doo21\]](#) for an overview). For a function f on a (not necessarily doubling) measure space (X, μ) taking values in some space E endowed with an extended norm (see [5.4](#)), we define its weak L^p norm to be

$$(4) \quad \|f\|_{L^{p,\infty}} := \sup_{t>0} t \cdot \mu(\{x \in X : t < |f(x)|\})^{\frac{1}{p}}$$

if $0 < p < \infty$, and $\|f\|_{L^{\infty,\infty}} := \|f\|_{L^\infty}$. The function f is said to be in weak L^p (`MemWLp`) if f is μ -a.e. strongly measurable with respect to a fixed topology on E (see e.g. [\[Doo21; Gou22\]](#) for motivation of the different notions of measurability in `mathlib`) and $\|f\|_{L^{p,\infty}} < \infty$. An important consequence is that if f is in weak L^p , then $|f|$ is finite almost everywhere (see `MemWLp.ae_ne_top`).

Given two measure spaces (X, μ) and (Y, ν) , and two topological spaces E and F endowed with extended norms, we say that an operator $T : (X \rightarrow E) \rightarrow (Y \rightarrow F)$ is of weak type (p, p') if there exists a constant C such that for any $f \in L^p(X; E)$, we have that Tf is ν -a.e. strongly measurable and

$$(5) \quad \|Tf\|_{L^{p',\infty}} \leq C \cdot \|f\|_{L^p}.$$

2.2. Statement of the main results.

- Statement of Theorem 1.0.2.
- Main propositions in Chapter 2.

3. PROJECT ORGANIZATION

- Project structure and division of tasks.
- Floris responsible for lemma statements (pros and cons).
- Interaction with harmonic analysis group.
- Contributors and roles.
- ToMathlib directory and contribution standards.
- Communication channels: Zulip, GitHub, Blueprint.

4. WORKING WITH A BLUEPRINT

4.1. The blueprint writing process.

- Blueprint writing process and finitary arguments.

4.2. **Changes and refinements.** Throughout the formalisation process, most of the blueprint only needed minor changes, such as fixing typos or clarifications of minor details. Let us highlight four aspects where formalisation resulted in more substantial changes or refinements.

4.2.1. *$I \leq J$ vs. $I \subset J$.* To be written!

4.2.2. *Just one top cube.* To be written!

4.2.3. *The Hardy–Littlewood maximal function.* One main ingredient in the proof is a Vitali covering argument, using the Hardy–Littlewood maximal function. To recall: given $f: \mathbb{R}^d \rightarrow \mathbb{C}$, the Hardy–Littlewood maximal function Mf of f is usually defined as

$$Mf: \mathbb{R}^d \rightarrow \overline{\mathbb{R}_{\geq 0}}, \quad x \mapsto \sup_{r>0} \frac{1}{|B(x,r)|} \int_{B(x,r)} |f(y)| dy.$$

If f is integrable, then Mf is (weak) L^1 , and hence finite almost everywhere. Initially, the blueprint only used a variant taking a supremum over finitely many balls: this ensured the resulting function was *always* finite, hence real-valued. Applying it to a countable collection of balls required taking a limit and using the monotone convergence theorem. This is a somewhat inelegant solution: most results about the maximal function do hold for any suprema, hence should be proven in the appropriate generality. We also prefer to avoid a superfluous limiting argument.

For the formal proof, we changed the definition to allow arbitrary suprema: this means the maximal function is $[0, \infty]$ -valued. Being able to speak of L^p -functions valued in $[0, \infty]$ prompted the introduction of *extended norms*, see Subsection 5.4.

4.2.4. *The real interpolation theorem.* The Marcinkiewicz real interpolation theorem is a standard result in functional analysis. It is also a key prerequisite for the proof of Carleson's theorem: for example, it is applied to the Hardy–Littlewood maximal function to deduce it is of strong type L^p , for $p \in (0, \infty)$. Our formalised version aims to reduce superfluous assumptions commonly found in the mathematical literature. For instance, it applies to any sub-additive function (not merely sub-linear ones) — that is, there is no need for a scalar multiplication on the target. Similarly, any positive exponent p is allowed (as opposed to demanding $p \geq 1$).

In light of generalising the Hardy–Littlewood maximal function, this proof had to be generalised again: applying it to the maximal functions requires a version which applies to functions with co-domain $[0, \infty]$ (or, more generally, any commutative monoid with a compatible norm). In return for a significant amount of refactoring work⁴, we have formalised a more general argument that is also conceptually clearer: for instance, it demonstrates that the subtractive structure on the co-domain is not used in a meaningful way. For further detail, we refer the reader to a separate article [PR].

4.3. **Constant tweaking.** To be written!

4.4. **Dealing with mistakes.**

- Lemma 11.1.6.
- Lemma 6.3.3/4.
- Hölder cancellation (radius $R \rightarrow 2R$).
- Lemma 6.2.3: additional hypothesis.

4.5. **Lessons learned.**

- Impact of blueprint choices on errors.
- Lack of definition environments \Rightarrow absence from dependency graph.

5. DESIGN DECISIONS

5.1. **Treatment of constants.** In analysis papers, one often needs to compare the growth of two functions $f, g : X \rightarrow Y$, where Y is a normed space such as the real numbers or the complex numbers. This is often expressed by statements of the form

$$(6) \quad \exists C > 0, \forall x \in X, \|f(x)\| \leq \|g(x)\|.$$

In many cases, one only cares about the existence of the positive constant C , but not about its concrete value. The constant can even be hidden in the notation, by writing (6) as

$$\forall x \in X, \|f(x)\| \lesssim \|g(x)\|.$$

This was also the case in the first version of [Bec+24a]. However, while Lean allows one to work with existential statements such as (6), this would have made the formalization harder. From the mathematical point of view, it would require the formalizers to figure out how large these constants needed to be and how the constants required to make each lemma hold were related to each other. From the technical side, it would require to work with witnesses for these existential statements, which can introduce some overhead. Therefore, while they were writing the blueprint, the authors of [Bec+24a] decided to make all of the constants explicit. At the beginning of the blueprint, they fixed a real number $1 < q \leq 2$ and a natural

⁴the initial formalisation was about 5000 lines long; the generalisation touched about 2000 of them

number $a \geq 4$, which is used to define the constants $D = 2^{100a^2}$, $\kappa := 2^{-10a}$ and $Z := 2^{12a}$. The bounding constants that appear in the blueprint theorem statements are expressed as functions of these fundamental constants.

In the formalization, we introduced a constant c with a value of 100, coming from the exponent of D . Every other constant is defined in terms of a , q and c , and we use the notation convention $\mathbf{Cx_y_z}$ to denote the constant appearing in result $\mathbf{x.y.z}$ of the blueprint.

As a side effect of the formalization project, once the main result had been formalized we were able to check that, replacing $D := 2^{100a^2}$ by 2^{7a^2} , the results still hold without needing to make any adjustments to the proofs. This improves the constant appearing in the main theorem by an order of magnitude.

5.2. Standing assumptions and the ProofData pattern. Mathematical papers often contain standing assumptions, that are in place for all the statements in the paper, or in a specific section. This avoids tedious repetition in the statements, sometimes at a slight expense in readability as the reader has to locate the places where such standing assumptions are made. Such a practice is especially important when the standing assumptions are lengthy and technical.

The standing assumptions for the statement of the generalization of Carleson’s theorem are quite formidable. We will not repeat them here, see [Bec+24b, Theorem 1.1.1] for details. Let us just mention that they involve a metric space X , a measure μ on X which is doubling for a constant 2^a , a family Θ of functions from X to \mathbb{R} , a distance on Θ satisfying five properties as well as a cancellativity condition, and a kernel $K: X \times X \rightarrow \mathbb{R}$ with Calderón–Zygmund like properties. The proofs involve even more additional data, notably two bounded measurable sets F and G , an integrability exponent $q \in (1, 2]$ and two measurable functions σ_1 and σ_2 with finite range.

One possibility to solve this issue in the formalization would be to say that there is no issue, and include as assumptions in each lemma the minimal subset of the standing assumptions that is necessary for the statement to make sense. This would come at a high readability cost. Moreover, each time one applies such a lemma one would need to provide the individual assumptions as a long list of parameters to the lemma.

It is a better idea to try to import in the formalization the practice of standing assumptions, in some form. One option would be to define a structure containing all these standing assumptions, and have it as a parameter in every lemma and every definition. The readability cost would be minor, but still there. In the Carleson project, we have used another strategy, making all the standing assumptions completely implicit — in particular, we do not need to pass them between lemmas. The implementation takes advantage of a feature of the language, typeclasses, initially designed for a completely different purpose.

The way to declare that a space is a metric space is the following line:

```
variable {X : Type*} [MetricSpace X]
```

The declaration between square brackets is the typeclass. When one writes the distance `dist x y` between two points x and y of X , the system tries to locate such a metric space typeclass on X , and uses the distance provided by this typeclass. The same system is used to provide algebraic operations on types, and properties of these.

Typeclasses can be used to capture standing assumptions as follows. Let us declare a new typeclass, of the form `[KernelProofData X]`, containing not a distance as in the above example, but a measure on X , a doubling constant, a family Θ of functions from $X \times X$ to \mathbb{R} ,

and so on, i.e., all our standing assumptions. Then, whenever one tries to use an object from the standing assumptions, the systems looks for a `KernelProofData` instance and fetches the object from the typeclass instance, just like it fetches the distance in a metric space.

Another advantage of this approach is that typeclasses can extend each other (just like a normed space contains more information than a metric space). This means we can craft one typeclass containing the standing assumptions for the statements, and another extended typeclass `ProofData` with more objects for the proofs. When a `ProofData` instance is in scope but Lean needs a `KernelProofData` instance, typeclass inference automatically fetches the latter from the former, just as it transparently interprets a normed space as a metric space.

The specific implementation of this idea in the Carleson project is slightly more involved, as it is written in the form `[ProofData a q K sigma1 sigma2 F G]`. However, all the additional data a , q , K , σ_1 , σ_2 , F and G are associated automatically and uniquely to X (in technical terms, they are *outparams*). This means that the user never has to give one of them explicitly, as we expect of a suitable system to handle standing assumptions.

5.3. Working with real numbers. The project differentiated between three different types of (extended) real numbers, \mathbb{R} , $\mathbb{R}_{\geq 0}$ and $\overline{\mathbb{R}_{\geq 0}}$, which gave necessary benefits but also created difficulties in the formalization and tension in design decisions. Indeed, in which of the three types to put constants, subsets, integrands, and perform computations?

A term x of type $\mathbb{R}_{\geq 0}$ is not also of type \mathbb{R} , nor is it of type $\overline{\mathbb{R}_{\geq 0}}$, even though in set-based mathematics there are inclusions of the corresponding sets. In type-theory however, one needs coercions, or casts, to go from one type to another. These coercions are often inserted and applied automatically. The `norm_cast` tactic can be really useful to normalize such casts.

However, one also needs conversions in the other directions: sometimes one would need to convert real numbers to nonnegative real numbers. The conversions often play together with other operations, in that one for instance would need to know if, and often under which conditions, the product of the conversions is the conversion of the product. Indeed, since `Real.toNNReal` maps negative numbers to zero, we get that `Real.toNNReal (-2) * Real.toNNReal (-3)`. Even with the necessary side conditions, the `norm_cast` tactic does not get very far in these situations. This can make conversions between number types quite painful, and puts some pressure to limit conversions and put as many numbers as possible in the same number type.

Sometimes one is forced to add a conversion, because certain operations are not defined on all types. Consider for instance the example of performing a computation on exponents in L^p -function spaces. Around the real interpolation theorem, one needs to compute p in terms of other exponents that naturally live in $\overline{\mathbb{R}_{\geq 0}}$. Yet there is no definition for a^p if p is in $\overline{\mathbb{R}_{\geq 0}}$, and therefore conversions from $\overline{\mathbb{R}_{\geq 0}}$ to \mathbb{R} are unavoidable.

There are also advantages of differentiating between the spaces. An advantage of putting for instance a constant in `NNReal` over `ENNReal` or `Real`, is that it encodes that the constant cannot be infinity and that it needs to be positive.

Another advantage of differentiating comes from the different levels of tactic support. For real numbers, computations are much easier to perform, using the `linarith` and `field_simp` tactics. These tactics do not work for terms in $\overline{\mathbb{R}_{\geq 0}}$ or $\mathbb{R}_{\geq 0}$. The tactic `ring` *does* work for these types, but not as well, and the performance for terms in $\overline{\mathbb{R}_{\geq 0}}$ is even worse than for terms in $\mathbb{R}_{\geq 0}$. For both $\overline{\mathbb{R}_{\geq 0}}$ and $\mathbb{R}_{\geq 0}$, it will generally fail when a goal needs properties of subtraction: it even fails to prove that $x - x = 0$. If $x : \overline{\mathbb{R}_{\geq 0}}$, the `ring` tactic can prove that $(x^{-1})^2 = (x^2)^{-1}$, or that $3/x = 3x^{-1}$, but it fails if $x : \mathbb{R}_{\geq 0}$. Similarly `norm_num` also works

for numbers in $\overline{\mathbb{R}_{\geq 0}}$, but again not as well as for real numbers, as even with the necessary side conditions it would not simplify a term of the form a/a . Both `ring` and `norm_num` fail to solve $(3/5) * (5/3) = 1$ when these numbers are viewed as numbers in $\overline{\mathbb{R}_{\geq 0}}$.

As a final example around considerations on which number types to use, consider the supremum of a set. As one eventually is anyway interested in situations in which this supremum is finite and the set is bounded from above, one could consider a setup in which the sets considered are subsets of real numbers and the suprema are real numbers as well. This was the choice at the outset of the project. However, in this setup, one needs to constantly carry around a proof or assumption that the set is indeed bounded. This is cumbersome, and therefore it turned out to be much better to let suprema just take values in the `ENNReals`.

Gradually, we came to the conclusion that, with some exceptions, it was best to use $\overline{\mathbb{R}_{\geq 0}}$ as much as possible.

5.4. Use of `ENorm`. This project motivated the creation of a new formalisation abstraction: *extended norms* (`enorms` for short) generalise many results from normed spaces to $\overline{\mathbb{R}_{\geq 0}}$. Their introduction was motivated by defining the Hardy–Littlewood maximal with co-domain $\overline{\mathbb{R}_{\geq 0}} = [0, \infty]$ (see Subsection 4.2). Previously, a statement “ $f \in L^p$ ” was only defined for functions $f: X \rightarrow E$ from a measure space X to a normed space E . Speaking about the maximal function being in weak L^p requires a notion of L^p functions with target $\overline{\mathbb{R}_{\geq 0}}$. Fortunately, many basic facts about L^p functions do not require the target to be a normed space and have reasonable analogues for $\overline{\mathbb{R}_{\geq 0}}$: for instance, the L^p norm of $f: X \rightarrow \overline{\mathbb{R}_{\geq 0}}$ for $p < \infty$ can be defined as $(\int_X |f(x)|^p dx)^{1/p}$, where the integrand is the function $X \rightarrow [0, \infty], x \mapsto |f(x)|^p$.

Motivated by this observation, we introduced a new notation class `ENorm`, describing a type endowed with a function `enorm: X → $\overline{\mathbb{R}_{\geq 0}}$` . This captures both normed spaces and $\overline{\mathbb{R}_{\geq 0}}$. The `enorm` of a normed space is the ambient norm, considered as a function into $\overline{\mathbb{R}_{\geq 0}}$; the `enorm` on $\overline{\mathbb{R}_{\geq 0}}$ is the identity function. Some definitions only require the existence of an `enorm`; many results require suitable compatibility conditions. Most lemmas necessary in this project only require an `ENormedAddCommMonoid`, i.e. an abelian monoid endowed with a continuous `enorm` that is positive definite and satisfies the triangle inequality. Many lemmas, in fact, only require slightly weaker assumptions.

Pursuing this approach required generalising all necessary definitions to `enorms`, as well as all basic results that are required in this project. This was a significant effort, but the verified nature of formalisation helped a lot by allowing to do so incrementally: in a first step, the new definition was added (together with the `enorm` instance on normed spaces). Then, all basic definitions were changed to allow an `enorm` as codomain, whenever this did not require further changes. A third step modified many lemma statements to use `enorms` instead of norms. The final phase involved generalising lemmas: often, this required small changes (such as, constants changing from a real to an extended real number, or adding a hypothesis about some quantity in $\overline{\mathbb{R}_{\geq 0}}$ being finite). At each point, verification ensured correctness: if all proofs still compiled, no more changes needed to be made.

To illustrate the magnitude of this refactoring, let us give some statistics: phase two (changing the basic definitions) changed about 300 lines of code; phase three (changing lemma statements to `enorms`) about 1000, and the lemma generalisations so far touched about 2500 lines of code.

The last phase (generalising mathlib lemmas) is not exhaustive: while substantial parts of mathlib have been generalised, some areas would require further refactor (and will follow as needed). To give an example, the total variation of a path in $\overline{\mathbb{R}_{\geq 0}}$ is a sensible quantity, even though $\overline{\mathbb{R}_{\geq 0}}$ is not a metric space. Introducing a weaker notion of extended metric spaces will enable new applications, and also provide the proper abstraction to state many scalar multiplication lemmas nicely. This is an ongoing effort by Felix Pernegger.

The process of generalisation revealed that this abstraction is also useful on its own: previously, many proofs in mathlib involved converting between norms taking values in \mathbb{R} or $\mathbb{R}_{\geq 0}$; using enorms provides a useful alternative. At the same time, it illustrates how the right abstractions can avoid duplicating work, while providing a useful clarification to mathematics.

5.5. Working with L^p functions. Working with L^p functions presents a design choice: should one work with members of L^p space (i.e., equivalence classes of functions up to almost everywhere equality, `MeasureTheory.Lp`), or work with representatives (`MemLp`) instead? The mathematical difference is very small: virtually all operations in the project respect a.e. equality. Formalization ergonomics, however, are greatly different: working with equivalence classes is much more cumbersome, as identities on the level of representatives only hold almost everywhere. (For example, $(f + g)x = fx + gx$ holds on the nose for actual functions; for representatives of \tilde{f}, \tilde{g} and $\tilde{f} + \tilde{g} \in L^p$, a priori this only holds for almost every x .) While the `filter_upwards` tactic simplifies reducing results from almost every to every x , these details add up: it is easiest to omit the passage entirely by working with functions in L^p (i.e., with `MemLp` directly).

Incidentally, lemmas in Mathlib are mostly stated for `MemLp` instead of `Lp`: this underscores this choice.

This relates well with the use of enorms (5.4): `Lp E p μ` as a normed space requires functions with codomain in a Banach space; this excludes e.g. $\overline{\mathbb{R}_{\geq 0}}$ which does not possess a well-behaved subtraction operation. Phrasing lemmas using `MemLp` first enables the generalization to $\overline{\mathbb{R}_{\geq 0}}$ in the first place.

This is somewhat similar to the trade-off between bundled and unbundled objects: established common wisdom is that unbundled or partially bundled objects are often better to work with than bundled ones. For instance, a continuous function $f: X \rightarrow Y$ would usually be formalized as `{f : X → Y}` (`hf : Continuous f`) instead of the bundled `f : ContinuousMap X Y`.

5.6. Test function classes. A common task during the formalization was to prove the various integrability and measurability sidegoals arising when estimating integrals. As is typical in harmonic analysis, most technical estimates are *a priori* estimates involving input functions in suitable test function classes.

In many parts of this project, bounded measurable compactly supported functions provided a convenient test function class. We also used the slightly larger class of bounded measurable functions supported on a set of finite measure in places where this weaker hypothesis matches the blueprint more closely. Since most constructions appearing in the proof leave these function classes invariant, integrability and measurability considerations are usually mathematically trivial.

It is desirable for this simplicity to be reflected in the formalization. This motivated packaging the recurring side conditions into test function classes.

The structures `BoundedCompactSupport` and `BoundedFiniteSupport` record, respectively, essentially bounded measurable functions with compact support and with support of finite measure. Once these hypotheses are bundled, later proofs can invoke short API lemmas instead of repeatedly reproving the same measure-theoretic facts. For example, from a hypothesis `hf : BoundedCompactSupport f` one immediately obtains that `f` is a member of any desired L^p space, or that `s.indicator f` again has bounded compact support when `s` is measurable. Lemmas such as `hf.carlesonSum` show that the relevant operators preserve the test function class, so once compact support has been established at the input, it remains available throughout the argument.

Further API extensions and better support from `fun_prop` could streamline these arguments even more.

5.7. Common pitfalls.

- Using `Real`.
- `Set.indicator` vs. `Measure.restrict`.
- `Finsets` vs. `Sets` in a `Fintype`.

6. CONCLUSION

- Project statistics (e.g. size of `ToMathlib` and total project).
- Summary of lessons learned:
 - Refer to general results early on.
 - Generalize during blueprint writing.

REFERENCES

- [Bec+24a] Lars Becker, Floris van Doorn, Asgar Jamneshan, Rajula Srivastava, and Christoph Thiele. “Carleson operators on doubling metric measure spaces”. arXiv:2508.05563 [math.CA]. 2024.
- [Bec+24b] Lars Becker et al. “A blueprint for the formalization of Carleson’s theorem on convergence of Fourier series”. <https://arxiv.org/abs/2405.06423>. 2024.
- [Doo21] Floris van Doorn. *Formalized Haar Measure*. 2021. arXiv: 2102.07636 [cs.LG]. URL: <https://arxiv.org/abs/2102.07636>.
- [Gou22] Sébastien Gouëzel. “A Formalization of the Change of Variables Formula for Integrals in mathlib”. In: *Intelligent Computer Mathematics*. Ed. by Kevin Buzzard and Temur Kutsia. Cham: Springer International Publishing, 2022, pp. 3–18. ISBN: 978-3-031-16681-5.
- [PR] Jim Portegies and Michael Rothgang. “Formalization of the real interpolation theorem for the Carleson project”. Manuscript, in preparation.

LARS BECKER, UNIVERSITY OF BONN
Email address: becker@math.uni-bonn.de

MARÍA INÉS DE FRUTOS-FERNÁNDEZ, UNIVERSITY OF BONN
Email address: midff@math.uni-bonn.de

LEO DIEDERING, UNIVERSITY OF BONN
Email address: leo.diedering@uni-bonn.de

FLORIS VAN DOORN, UNIVERSITY OF BONN
Email address: vdoorn@math.uni-bonn.de

SÉBASTIEN GOUËZEL, UNIVERSITY OF RENNES
Email address: sebastien.gouezel@univ-rennes1.fr

EVGENIA KARUNUS, UNIVERSITY OF BONN
Email address: lakesare@gmail.com

EDWARD VAN DE MEENT, UNIVERSITY OF UTRECHT
Email address: edwardvdmeent@gmail.com

PIETRO MONTICONE, UNIVERSITY OF TRENTO
Email address: pit.monticone@gmail.com

JASPER MULDER-SOHN, THE HAGUE
Email address: jasper.mulder@planet.nl

JIM PORTEGIES, EINDHOVEN UNIVERSITY OF TECHNOLOGY
Email address: j.w.portegies@tue.nl

JORIS ROOS, UNIVERSITY OF MASSACHUSETTS LOWELL
Email address: joris_roos@uml.edu

MICHAEL ROTHGANG, UNIVERSITY OF BONN
Email address: rothgang@math.uni-bonn.de

JAMES SUNDSTROM, BARUCH COLLEGE
Email address: james.sundstrom@baruch.cuny.edu

JEREMY TAN, NATIONAL UNIVERSITY OF SINGAPORE
Email address: jtjierui@comp.nus.edu.sg