THE CONTINUOUS FUNCTIONAL CALCULUS IN LEAN

ANATOLE DEDECKER AND JIREH LOREAUX

ABSTRACT. The continuous functional calculus is perhaps the most fundamental construction in the theory of operator algebras, especially C^* -algebras. Here we document our formalization of the continuous functional calculus in Lean, which constitutes the first such formalization in any proof assistant. Our implementation is already merged into Lean's mathematical library, Mathlib. We provide a brief introduction to the mathematical theory for those unfamiliar with the subject, and then highlight the design decisions in our formalization which proved to be important for usability. Our exposition is aimed at a general mathematical audience and provides a glimpse into the world of formalization by laying bare the discovery process.

1. Introduction

Every mathematical discipline has a toolbox which is invaluable to its practitioners, is used to lay the foundations for the subject and becomes such an essential technique that it is often taken for granted. In the theory of C^* -algebras, the continuous functional calculus is the first such tool. This functional calculus identifies certain commutative C^* -subalgebras with algebras of continuous functions on a compact Hausdorff space, thereby unlocking the ability to reason about C^* -algebras using elementary arguments about continuous functions. Our goal in this paper is to document our formalization of the continuous functional calculus in Lean in such a way that the reader can understand the motivating factors leading to this design among the multitude of possibilities.

This paper is intended to be accessible to a relatively wide mathematical audience. In particular, we hope it is accessible both to C^* -algebraists with little background in formalization, and also to those familiar with formalization in Lean with no background in operator algebras. We begin in Section 2 with a brief overview of the mathematical theory of the continuous functional calculus targeted at general mathematical audience, and then discuss in Section 3 the design considerations and litmus tests which guided our formalization.

In Section 4 we walk leisurely through the design space of the continuous functional calculus in the most elementary setting — complex-valued continuous functions on the spectrum of a normal element in a unital C^* -algebra — accentuating the importance of usability. This presentation, while not a precise representation of the actual development process, reflects some of the approaches we considered and evaluates their short-comings. This is intended to provide the reader, especially a mathematician not fluent in formalization, with a sense of how and why implementation in formalization matters, the questions that need to be considered, as well as how these may be satisfactorily resolved.

Section 5 piggybacks on the previous section to introduce the class-based design of the continuous functional calculus (still for $unital\ C^*$ -algebras), presented as a sequence of attempts converging iteratively to the version implemented in Mathlib¹ [mC20]. While the focus of Section 4 is primarily usability, Section 5 is concerned instead with generality and flexibility.

In Section 6 we discuss the uniqueness of the continuous functional calculus, and why it is a separate class and the benefit it provides. We highlight, in terms of a few examples, a key difference in the approach to proofs utilizing uniqueness often used on paper versus in Lean. In particular, on paper, many mathematicians make direct appeal to polynomials and the Stone–Weierstrass theorem, whereas in Lean we prefer to utilize uniqueness directly. Perhaps surprisingly to the uninitiated, we develop a continuous functional calculus over $\mathbb{R}_{\geq 0}$ (the semifield of nonnegative real numbers). This has many benefits, but also contributes some unavoidable mathematical difficulties. We overcome some of them with our implementation of the uniqueness class for the continuous functional calculus, but we highlight some other difficulties in Section 11.

²⁰²⁰ Mathematics Subject Classification. 68V20, 47B15, 46L05, 68V35.

 $[\]textit{Key words and phrases.}$ continuous functional calculus, C^* -algebras, Lean, Mathlib.

¹A copy of this implementation is included in the code repository associated to this paper.

As C^* -algebraists will know, non-unital algebras are an absolutely essential part of the theory. As such, our formalization would be remiss if it did not include these. While the implementation largely follows the same pattern as for unital algebras, there are a few nontrivial considerations addressed in Section 7.

Section 8 provides a brief description of the techniques we use to provide instances of all the continuous functional calculus variations we have implemented. This is somewhat interesting in its own right, because we construct instances of the non-unital calculus for C^* -algebras by means of the existing instances of the unital calculus, rather than a direct appeal to the Gelfand transform.

Automation is a key piece of our interface, and it is crucial to minimizing distraction to, and maximizing the productivity of, the end user, which we describe in Section 9. In the code artifact associated to this paper, we give example implementations of the tasks proffered in Section 3 as litmus tests to evaluate the usability of our design, thereby providing examples of our interface in action and highlighting the automation.

In Section 10 we describe a variation of our continuous functional calculus class encoding that the homomorphism is isometric, which is intentionally omitted from earlier versions. We conclude in Sections 11 and 12 with a discussion of the current limitations of our approach and some ideas for future work.

2. Mathematical background

A C^* -algebra \mathcal{A} is a complex² Banach *-algebra (i.e., $x \mapsto x^*$ is a conjugate-linear antimulitplicative involution which distributes over scalar multiplication) whose norm satisfies the C^* -identity: $\|a^*a\| = \|a\|^2$ for any $a \in \mathcal{A}$. The * operation is called the *adjoint* because a canonical example of a C^* -algebra is the algebra of $n \times n$ matrices over \mathbb{C} equipped with the operator norm induced by the standard Euclidean norm. More generally, one may consider the bounded linear operators on a complex Hilbert space, and in fact, every C^* -algebra is isomorphic to a norm-closed *-subalgebra of the bounded operators on some Hilbert space [Dav96, Theorem I.9.12]. The standard *commutative* example of a C^* -algebra is the algebra $C(X, \mathbb{C})$ of continuous complex-valued functions on a compact Hausdorff space X, equipped with the supremum norm where * is given by pointwise conjugation.

A C^* -algebra may either contain an identity element or not. In the literature, in the former case, \mathcal{A} is said to be unital. Without that adjective, the algebras in question may or may not be unital; sometimes the literature uses the adjective non-unital to emphasize that \mathcal{A} explicitly does not have an identity, but usage varies. In contrast, Mathlib prefers to assume algebraic structures contain an identity element unless otherwise specified; when an algebraic structure may or may not contain an identity, the prefix NonUnital is added. We will adopt Mathlib's convention in this paper.

A key concept in the theory of C^* -algebras is that of the spectrum of an element $a \in \mathcal{A}$, which is defined as:

$$\sigma_{\mathbb{C}}(a) := \{ \lambda \in \mathbb{C} \mid \lambda 1 - a \text{ is not invertible} \}$$

On its face, this definition only makes sense for unital algebras³.

However, for an element of a non-unital C *-algebra A, one can still consider its spectrum in some unital C *-algebra containing A, typically the minimal such algebra. Additionally, because of *spectral permanence* [Tak10, Proposition 4.8], this doesn't depend on the choice of such a unital algebra, which makes this object quite useful and interesting.

Sometimes the theory of C^* -algebras is described as noncommutative topology. The reasoning behind this terminology is that $Gelfand\ duality$ [Gel41, Neg71] establishes a contravariant equivalence between the category of commutative unital C^* -algebras and the category of compact Hausdorff spaces⁴. The contravariant functors involved in this duality are the maps $X \mapsto C(X, \mathbb{C})$ taking a compact Hausdorff space to the C^* -algebra of continuous complex-valued functions, and $A \mapsto \hat{A}$ taking a commutative C^* -algebra to its character space (somewhat confusingly, this is also called the spectrum, although there is a connection; it

²There also exists a theory of real C^* -algebras, but for the most part it is outside the scope of this paper, although we make brief mention of it again in Section 12. Those familiar with formalization might be surprised that one would begin with the formalization of the continuous functional calculus before the formalization of real C^* -algebras, but it is important to realize that the real theory actually depends heavily on the complex theory. In fact, the category of real C^* -algebras is equivalent to the category of complex C^* -algebras equipped with an extra conjugate-linear multiplicative involution.

³Not only that, but the reader may be surprised to see that Mathlib only requires the scalars R to be a commutative semiring, while A must be a ring. This is a minor quirk that we exploit to great benefit later.

can also be called the *maximal ideal space*), which consists of the nonzero algebra homomorphisms from \mathcal{A} into \mathbb{C} . The natural isomorphisms involved in this duality are the Gelfand transform and point evaluation.

The continuous functional calculus is a part of the standard toolbox in C^* -algebra theory that exploits this connection. Given an element $a \in \mathcal{A}$, we say a is normal if it commutes with its adjoint: $a^*a = aa^*$. The C^* -algebra generated by a normal element a is commutative, and so by the duality above, it is isomorphic to $C(X,\mathbb{C})$ for some compact Hausdorff space X. In fact, X is naturally identified with $\sigma_{\mathbb{C}}(a)$, the spectrum of a, and moreover, the identity function is mapped to a under this isomorphism. One therefore obtains a (necessarily isometric) *-monomorphism from $C(\sigma_{\mathbb{C}}(a),\mathbb{C})$ into \mathcal{A} that sends the identity function to a, and this is referred to as the continuous functional calculus. In the literature, evaluation of the continuous functional calculus of a normal element a at a function $f \in C(\sigma_{\mathbb{C}}(a),\mathbb{C})$ is denoted $f(a)^5$. Finally, by the Stone–Weierstrass theorem, this is the unique continuous *-homomorphism from $C(\sigma_{\mathbb{C}}(a),\mathbb{C})$ which extends the natural homomorphism of the polynomials $\mathbb{C}[z]$ on a.

However, there is yet more to be said. There is the *spectral mapping theorem*: if a is normal and f is a continuous function on $\sigma_{\mathbb{C}}(a)$, then $\sigma_{\mathbb{C}}(f(a)) = f(\sigma_{\mathbb{C}}(a))$. Perhaps the most crucial property of the continuous functional calculus is:

Theorem (Composition property). If a is normal and f is a continuous function on $\sigma_{\mathbb{C}}(a)$, and g is continuous on $\sigma_{\mathbb{C}}(f(a))$, then $g \circ f$ is continuous on $\sigma_{\mathbb{C}}(a)$ and $(g \circ f)(a) = g(f(a))$.

The notational simplicity belies its nontrivial mathematical content: on the left-hand side is the evaluation of the continuous functional calculus of a at $g \circ f$, whereas on the right-hand side is the evaluation of the continuous functional calculus of f(a) (which is normal) at g. This property depends crucially on the fact that the continuous functional calculus for a given normal element a is unique, which is a straightforward consequence of the Stone–Weierstrass theorem.

In addition, if $a \in \mathcal{A}$ is selfadjoint (i.e., $a^* = a$), then the spectrum of a is contained in \mathbb{R} . For such elements, the C^* -algebraist often restricts attention to real-valued functions of a real variable, in which case the outputs are also selfadjoint. Likewise, if $a \in \mathcal{A}$ is nonnegative (i.e., $a = b^*b$ for some $b \in \mathcal{A}$), then the spectrum of a is contained in $[0,\infty)$ (herein denoted $\mathbb{R}_{\geq 0}$), and one may often restrict attention to nonnegative functions. The considerations of the previous two paragraphs allow, for instance, to define the square root of a nonnegative element $a \in \mathcal{A}$ and to conclude that $\sqrt{a^2} = a$.

The interplay between unital and non-unital C^* -algebras is a deep and important part of the subject, and ensuring non-unital C^* -algebras are on equal footing with unital ones is an essential aspect of its formalization. Although it may seem backwards to those unfamiliar with C^* -algebra theory, the non-unital theory is often developed as a consequence of the unital theory. This makes more sense when one considers the commutative case and Gelfand duality, where the category of unital (respectively, non-unital) commutative C^* -algebras is contravariantly equivalent to the category of (respectively, pointed) compact Hausdorff spaces.

3. Design considerations

In formalization, especially for constructions or theorems that experience frequent use, it is often the case that the seemingly most natural statement is not the one that is dictated by utilitarian design. In this section we outline the considerations that guided the design of our interface to the continuous functional calculus in Lean, along with a few tests used to determine its suitability. For the sake of brevity, in this section we will not delve into the meaning of jargon like *unbundling* or *rewriting*. Readers unfamiliar with formalization may wish to skim this section and trust that the relevant ideas and reasoning for these design constraints will be adequately explained *in situ* in the following sections.

- 3.1. **Requirements.** There are a number of design constraints which we placed upon ourselves in order to ensure that our interface would meet the needs of Mathlib users. Our design should:
 - (1) Allow for easy rewriting of terms involving the continuous functional calculus.
 - (2) Allow for easy use of unbundled functions, especially in cases where those functions are obviously continuous on the spectrum; for instance, when the spectrum is finite or discrete.

⁴The category of not-necessarily-unital commutative C^* -algebras is *not* equivalent to the category of locally compact Hausdorff spaces (with proper maps as morphisms), but rather to the category of pointed compact Hausdorff spaces [Del09].

⁵If this notation seems confusing, it's worth understanding the continuous functional calculus as an extension to continuous functions of the polynomial evaluation map $p \in \mathbb{C}[z] \mapsto p(a)$, from which this notation arises naturally.

- (3) Provide easy and straightforward use of the composition property (i.e., proving things like $f(-a) = (x \mapsto f(-x))(a)$ and $f(a^*) = (x \mapsto f(\bar{x}))(a)$ should be nearly trivial).
- (4) Ensure the full continuous functional calculus (i.e., the *-isomorphism with the subalgebra generated by the element) is recoverable from this more general version.
- (5) Allow for multiple scalar (semi)rings (namely, \mathbb{C} , \mathbb{R} and $\mathbb{R}_{\geq 0}$) with a unified interface. This allows users to work with the continuous functional calculus on selfadjoint or nonnegative elements by considering only functions $\mathbb{R} \to \mathbb{R}$ or $\mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, respectively.
- (6) Avoid requiring the algebra to be a metric space to allow for types that may not be equipped with a metric structure globally in Mathlib. The canonical use case here is matrices, which are intentionally not equipped with a metric structure in Mathlib so as to avoid choosing a preferred metric.
- (7) Allow for a non-unital continuous functional calculus, and obtain the non-unital calculus from the unital one when the algebra is unital.
- (8) Allow for future generalization to real C^* -algebras

Items 1 to 4 are all addressed to some extent in Section 4, although item 3 makes another appearance in Section 6. Items 4 to 6 are addressed in Sections 5 and 6. Section 12 contains a brief discussion of item 8, although it is largely outside the scope of this paper. Sections 9 and 10 are related to Item 6 and to items 1 and 2, respectively.

There are several tasks which the continuous functional calculus should be able to accomplish, and they require essentially no prerequisites. Before embarking on this project, we set for ourselves a series of litmus tests to evaluate whether our design was sufficiently usuable to be effective in practice. If these were exceedingly difficult to implement, then the design was not suitable. Example implementations of each, with the exception of the last one⁶, can be found in the code artifact associated to this paper.

Inverses: For an invertible element a and a function f which is continuous on the spectrum of the inverse of a, show that: $f(a^{-1}) = (x \mapsto f(x^{-1}))(a)$. This test requires:

- (1) A continuous functional calculus over \mathbb{C} for normal elements.
- (2) Use of functions which are not everywhere continuous (i.e., $x \mapsto x^{-1}$).
- (3) Application of the composition property (since we are working with continuous functional calculi for both a and a^{-1}).

Positive and negative parts: For a selfadjoint element a in a non-unital C^* -algebra, there are commuting positive elements a^+ and a^- such that $a = a^+ - a^-$ and $a^+a^- = 0$. This test has slightly different requirements:

- (1) A (non-unital!) continuous functional calculus over \mathbb{R} for selfadjoint elements.
- (2) Use of auto-params involving the condition that f(0) = 0.

Square roots: For a nonnegative element a in a non-unital C^* -algebra, there is a unique nonnegative element $a^{\frac{1}{2}}$ so that $(a^{\frac{1}{2}})^2 = a$. This requires:

- (1) A (non-unital!) continuous functional calculus over $\mathbb{R}_{>0}$ for nonnegative elements.
- (2) Application of the composition property, which requires uniqueness of the functional calculus. This is trickier over $\mathbb{R}_{>0}$ than over \mathbb{R} or \mathbb{C} because Stone–Weierstrass doesn't apply.
- (3) Showing that we can use different functional calculi (i.e., over \mathbb{R} too!)

Matrices over \mathbb{R} : Is it possible to create an instance of the continuous functional calculus on selfadjoint elements of Matrix n n \mathbb{R} , i.e., $n \times n$ matrices over \mathbb{R} ? The primary reason to include this test is that it requires the continuous functional calculus to work for algebras over \mathbb{R} without a metric structure.

4. An interface for unital C^* -algebras

The aim of this section is to motivate and describe the most fundamental design choices made in our formalization: unbundling, and the use of so-called " $junk\ values$ ". In order to keep that presentation simple and, hopefully, accessible to a broad mathematical audience with no background in formalization, we focus on the most basic setting, that of normal elements in $unital\ complex\ C^*$ -algebras, and we postpone the detailed exposition of more technical requirements to later sections. For that reason, we emphasize that the exposition in this section does not actually represent faithfully the design process, and should rather be understood as a motivation and description of some general principles.

⁶This has nevertheless been implemented in Mathlib already.

4.1. A mathematician's definition. The first step of our work was of course to formalize the mathematical content underlying the continuous functional calculus, namely the (unital) Gelfand transform and spectral permanence. In fact, these were added to Mathlib back in February, 2023⁷, together with the usual definition of continuous functional calculus described in Section 2. Although we won't focus on the formalization of the prerequisites, we recall that first definition of the functional calculus in Listing 1, which will serve as our starting point for the design process. Let us also use it to explain some glimpses of Lean syntax for the unfamiliar reader.

First, the variable line declares A to be an arbitrary unital C^* -algebra. In that context, we define a new term, i.e a mathematical object, named continuousFunctionalCalculus, which takes as input a normal element a of A, and outputs a term of $type\ C(spectrum\ \mathbb{C}\ a,\ \mathbb{C})\ \simeq\star_a[\mathbb{C}]$ StarAlgebra.elemental $\mathbb{C}\ a$. Here, the expression $C(spectrum\ \mathbb{C}\ a,\ \mathbb{C})$ denotes the type of continuous functions on the spectrum of a with values in \mathbb{C} , while StarAlgebra.elemental $\mathbb{C}\ a$ is a name for the the unital C^* -subalgebra generated by a, and the notation $\simeq\star_a[\mathbb{C}]$ refers to the type of *-isomorphisms between these *-algebras (over \mathbb{C}). The actual definition of the term continuousFunctionalCalculus a, expressed using the gelfandStarTransform of the C^* -algebra StarAlgebra.elemental $\mathbb{C}\ a$, comes after the symbol :=. Finally, we prove a theorem, stating that the continuous functional calculus for some normal element a maps (the restriction to the spectrum of a of) the identity function to a.

```
Listing 1: The continuous functional calculus as a *-isomorphism
   LeanCFC/Snippets/Baseline.lean
8 variable {A : Type*} [CStarAlgebra A]
10 /-- The *-isomorphism between the continuous functions on the spectrum of 'a'
11 and the (unital) C⋆-subalgebra generated by 'a'. -/
12 def continuousFunctionalCalculus (a : A) [IsStarNormal a] :
       \mathbb{C}(\text{spectrum }\mathbb{C} \text{ a, }\mathbb{C}) \simeq \star_a[\mathbb{C}] \text{ StarAlgebra.elemental }\mathbb{C} \text{ a } :=
13
14
     characterSpaceHomeo a ▷.compStarAlgEquiv' ℂ ℂ ▷.trans
       (gelfandStarTransform (elemental € a)).symm
15
16
17 /-- The continuous functional calculus identifies the (restriction of) the
18 identity function on the spectrum of 'a' with 'a'. -/
19 theorem continuousFunctionalCalculus_map_id (a : A) [IsStarNormal a] :
20
       continuousFunctionalCalculus a (.restrict (spectrum € a) (.id €)) =
         ⟨a, self_mem ℂ a⟩ :=
21
     gelfandStarTransform (elemental € a) ▷.symm_apply_apply _
```

While this definition is completely satisfying from a mathematical point of view, let us recall that the functional calculus is not the end of the story. Rather, it is a crucial stepping stone on the road towards the general theory of C^* -algebras, where it will be used extensively. Thus, to progress towards formalization of these later steps, it is essential to provide an interface dictated by utilitarian design rather than mathematical simplicity. Listing 2 presents such an interface, which we will explain, motivate and expand in the rest of this section. Although this is not yet the version that you can find in Mathlib, it is a good approximation: more precisely, in the context of unital complex C^* -algebras, cfc f a agrees with the one in Mathlib.

```
Listing 2: A more convenient interface for the continuous functional calculus LeanCFC/Snippets/Intermediate.lean

6 variable {A : Type*} [CStarAlgebra A]

7 s open scoped Classical in
9 def cfc (f : \mathbb{C} \to \mathbb{C}) (a : A) : A :=
10 if h : IsStarNormal a \land Continuous ((spectrum \mathbb{C} a).restrict f)
11 then
12 letI := h.1
13 continuousFunctionalCalculus a \land (spectrum \mathbb{C} a).restrict f, h.2\land
```

⁷https://github.com/leanprover-community/mathlib3/pull/17164

```
else 0
14
15
16 lemma cfc_def \{f: \mathbb{C} \to \mathbb{C}\}\ \{a: A\}\ [ha: IsStarNormal a]\ (hf: ContinuousOn f (spectrum <math>\mathbb{C} a)):
       cfc f a = continuousFunctionalCalculus a (_, hf.restrict) := by
     rw [continuousOn_iff_continuous_restrict] at hf
18
     simp [cfc, ha, hf]
19
20
21 lemma cfc_id (a : A) [ha : IsStarNormal a] :
       cfc id a = a := by
22
     rw [cfc_def continuousOn_id]
23
     exact congrArg _ (continuousFunctionalCalculus_map_id a)
24
```

4.2. Simple expressions. The distinction between Listing 2 and Listing 1 is completely elementary: we merely extend⁸ the definition of f(a) to any map $f: \mathbb{C} \to \mathbb{C}$ and any element $a \in A$, by letting f(a) = 0 whenever a is not normal or f is not continuous on $\sigma_{\mathbb{C}}(a)$. Mathematically, this is completely uninteresting, since none of the properties — not even linearity in the variable f! — extend without these hypotheses. Nevertheless, this kind of extension by "junk values" is a common practice in formalization, perhaps the most prevalent example being that Lean⁹ defines $(0:\mathbb{R})^{-1}:=0$.

The motivation behind this is that we want to emulate the "Write first, think later" practice of informal mathematics. On paper, it is very common to write down an entire, complicated computation, before checking (or letting the reader check) that every term is well-defined — e.g, each denominator is nonzero, each integrand is integrable, and so forth — and that the manipulations of these terms are legal. While Lean is prefectly fine with waiting a bit for a proper justification of some proof step, it has no tolerance for ill-defined expressions: after all, the core job of the Lean checker is precisely to check that each expression, whether it denotes a mathematical object or a proof, is well-formed. By extending the set of well-defined expressions using junk values, we replace ill-defined objects by ill-behaved ones, which means that the usual checks are delayed until we have to prove something nontrivial about these — e.g, $a^{-1} * a = 1$ only holds for $a \neq 0$.

Coming back to functional calculus, it may not be clear how this discussion applies to Listing 1. After all, the expression continuousFunctionalCalculus a (.restrict (spectrum \mathbb{C} a) (.id \mathbb{C})) does not seem to contain any proof of normality or continuity. In the case of continuity, this is because the expression .restrict (spectrum \mathbb{C} a) (.id \mathbb{C}) has type \mathbb{C} (spectrum \mathbb{C} a, \mathbb{C}), meaning that it comes bundled [Baa22, Section 6] with a proof of continuity. Unfortunately, this only works because Mathlib provides us with the relevant building blocks — the identity as an element of $\mathbb{C}(\mathbb{C}, \mathbb{C})$, and a restriction operation specific to continuous functions. By contrast, the expression $\exp(a)$ would be written as in Listing 3, where by fun_prop is a proof of continuity. Regarding normality, Listing 1 relies on typeclass inference: the square brackets around IsStarNormal a indicate to Lean that it should search for a normality proof using the instances declared in the library 10. While this is practical in basic cases, one cannot always rely on the typeclass system. This happens for example in the expression f(a+a) is since it is not true that the sum of two normal elements is normal, Lean doesn't know that f(a+a) is normal. Thus, to write down such an expression using Listing 1, one would need to first prove that normality — and register it with the typeclass system using have 12. This is demonstrated in Listing 4.

⁸One could argue that we also restrict our attention to functions defined on all of \mathbb{C} , but since we don't care about continuity outside of $\sigma_{\mathbb{C}}(a)$ this doesn't change anything mathematically.

⁹As well as most proof assistants.

 $^{^{10}}$ For example, Lean will be able to synthesize such an instance when the algebra A is commutative.

¹¹If that sounds like something no one would ever write, remember that it could just be a simple step in the middle of a complicated computation.

 $^{^{12}}$ To be complete, there are still ways to work around that by declaring carefully-chosen instances. Another way would be to compute a + a in the commutative C^* -subalgebra generated by a. See Listing 11 and the surrounding discussion for an example.

```
Listing 3: Definition of exp(a) using the approach of Listing 1

LeanCFC/Snippets/Baseline.lean

24 example {a : A} [IsStarNormal a] : StarAlgebra.elemental ℂ a :=
25 continuousFunctionalCalculus a ⟨fun x → Complex.exp x, by fun_prop⟩
```

```
Listing 4: Definition of f(a+a) using the approach of Listing 1 

LeanCFC/Snippets/Baseline.lean

27 example {a : A} [ha : IsStarNormal a] {f : C(spectrum \mathbb{C} (a + a), \mathbb{C})} :
28    StarAlgebra.elemental \mathbb{C} (a + a) :=
29    have : IsStarNormal (a + a) := by
30    rw [isStarNormal_iff, star_add]
31    exact (ha.star_comm_self.add_left ha.star_comm_self).add_right
32    (ha.star_comm_self.add_left ha.star_comm_self)
33    continuousFunctionalCalculus (a + a) f
```

Another convenient feature of Listing 2 is that it avoids the use of subtypes like $StarAlgebra.elemental \ C$ a or spectrum \ C a. To explain what we mean by "subtype" 13, we need to point out a key distinction between Lean's underlying theory and a more familiar set theory. In set theory, the \ e relation is nothing more than a binary relation on all mathematical objects (i.e sets). In particular, any set can (and does) belong to a lot of different sets, and there is no preferred one. By contrast, in Lean, each term has a unique type, hence there is no notion of one type being included in another. Instead, what you can do is declare canonical maps between types, called coercions or casts, which Lean uses to interpret elements of some type as elements of another type. Lean tries to insert these automatically, but one easily gets into situations where there is not enough context to do so. In these cases, the user would need to provide a hint (i.e., a type ascription) to help Lean determine it should insert the coercion, which greatly reduces the readability. On top of that, even when inserted automatically, Lean does keep track of these functions in the expressions, where they appear as \(\tau. This means one has to constantly use compatibility results like \(\gamma(a + b) = \gamma + \gamma b\) or \(\gamma = 0\), which tend to make computations really messy.

Likewise, the use of the type spectrum $\mathbb C$ a in Listing 1 implies that we constantly have to restrict ¹⁴ f to the spectrum of a. This pain point may already be observed in Listing 1 in the case of the identity function, and Listing 5 shows how this gets even worse when f is defined on another subtype which "contains" the spectrum — keep in mind these complicated expressions are just for being able to write f(a)! Of course, the approach in Listing 2 does suffer from the dual issue, in the sense that one may need to extend a function to be defined on all of $\mathbb C$. Here though, the common practice of extending functions using junk values works in our favor, since it means that essentially any function one might want to write down is already defined everywhere.

```
Listing 5: Troubles with restrictions : f(u) for f \in C(\mathbb{S}^1, \mathbb{C}) and u unitary LeanCFC/Snippets/Baseline.lean

39 example \{u: unitary A\} \{f: C(Metric.sphere (0 : C) 1, C)\} : A:=
40 continuousFunctionalCalculus \{u: A\}
41 \{f: C(Metric.sphere (0 : C) 1, C)\} : A:=
42 \{f: C(Metric.sphere (0 : C) 1, C)\} : A:=
43 \{f: C(Metric.sphere (0 : C) 1, C)\} : A:=
```

Before moving on, let's impart some nuance to the first comparison between Listing 1 and Listing 2. As stated, we have lost some significant mathematical content about the functional calculus: with our definition, $f \mapsto f(a)$ is neither surjective (because we forgot its image), nor injective (because it doesn't depend on the value of f outside of $\sigma_{\mathbb{C}}(a)$), nor even an algebra homomomorphism (because of junk values). More subtly, we lost the fact that it is isometric when restricted to $C(\sigma_{\mathbb{C}}(a), \mathbb{C})$, which is of crucial importance for the rest of the theory. Listing 6 shows how this can be fixed rather easily by restating all the relevant facts as

 $^{^{13}}$ Technically, what follows is a description of types equipped with a coercion, and subtypes mean a more precise thing, but this distinction won't matter to us.

 $^{^{14}}$ Note that restricting is the same as composing with a coercion.

separate lemmas in the language of Listing 2 (Let us note that the corresponding proofs are quite messy, precisely because we have to use the interface of Listing 1. We omit some of them for clarity).

Nevertheless, it is not clear at this stage that any of this is worth the trouble. First, setting up a huge interface for a specific construction does come with some disadvantages in terms of maintenance cost and steepening the learning curve; for example, a new user might try to use the generic map_add rather than the specific cfc_add . Furthermore, the use of bare functions instead of bundled continuous maps or bundled *-algebra homomomorphisms causes some friction when using the library. As a crucial example, working with the type $C(spectrum \ C \ a, \ C)$ has the inherent advantage that it is equipped with its natural C^* -norm, which we have to emulate to express the isometric property in Listing 6. In fact, our initial designs still used bundled continuous maps and *-algebra homomomorphisms. We will now see more profound arguments in favor of Listing 2.

```
Listing 6: Recovering some properties in the setting of Listing 2
   LeanCFC/Snippets/Intermediate.lean
41 lemma cfc_add \{f g : \mathbb{C} \to \mathbb{C}\} \{a : A\} [ha : IsStarNormal a]
        (hf : ContinuousOn f (spectrum ℂ a)) (hg : ContinuousOn g (spectrum ℂ a)) :
        cfc (f + g) a = cfc f a + cfc g a := by
43
     rw [Pi.add_def, cfc_def (hf.add hg), cfc_def hf, cfc_def hg]
44
45
     exact map_add (continuousFunctionalCalculus a) <_, hf.restrict> <_, hg.restrict>
46
48 lemma cfc_neg \{f : \mathbb{C} \to \mathbb{C}\}\ \{a : A\}\ [ha : IsStarNormal a]
        (hf : ContinuousOn f (spectrum € a)) :
49
50
        cfc(-f) a = -(cfc f a) := by
     rw [Pi.neg_def, cfc_def hf, cfc_def hf.neg]
51
     norm_cast
52
     exact map_neg (continuousFunctionalCalculus a) (_, hf.restrict)
53
54
   -- Similarly, we prove 'cfc_mul', 'cfc_sub', 'cfc_const'...
55
57 lemma cfc_eq_iff \{f g : \mathbb{C} \to \mathbb{C}\} \{a : A\} [ha : IsStarNormal a]
        (hf : ContinuousOn f (spectrum € a))
58
        (hg : ContinuousOn g (spectrum \mathbb{C} a)) :
59
        cfc f a = cfc g a \leftrightarrow \forall x \in \text{spectrum } \mathbb{C} a, f x = g x := by
60
     simp [cfc_def hf, cfc_def hg, ContinuousMap.ext_iff]
61
62
63 lemma range_cfc_eq {a : A} [ha : IsStarNormal a] :
64
        Set.range (fun f → cfc f a) = StarAlgebra.elemental C a :=
65
66
67 lemma norm_cfc_eq \{f : \mathbb{C} \to \mathbb{C}\}\ \{a : A\}\ [ha : IsStarNormal a]
        (hf : ContinuousOn f (spectrum € a)) :
68
        \|\mathsf{cfc}\ \mathsf{f}\ \mathsf{a}\| = \mathsf{sInf}\ \{\mathsf{C}\ :\ \mathsf{R}\ \big|\ 0 \le \mathsf{C}\ \land\ \forall\ \mathsf{x} \in \mathsf{spectrum}\ \mathbb{C}\ \mathsf{a},\ \|\mathsf{f}\ \mathsf{x}\| \le \mathsf{C}\} :=
69
70
     sorry
```

4.3. Rewriting with ease by avoiding dependent types. Following computer science terminology, rewriting refers to the act of replacing one expression e by an equal expression e', inside some larger statement or expression. In Lean this is accomplished by using the tactic rw (short for "rewrite"): if H is a proof of some equality a = b, rw [H] replaces all occurences of a by b in the goal or some other specified location. This tactic underlies essentially every computation in Lean, and as such one should expect to be very frequently rewriting either f or a in an application f(a) of the functional calculus.

Due to its fundamental nature, rewriting works out of the box in most settings: after all, what it *means* for two objects to be equal is that we can substitute one for another in any expression or statement, no matter how complicated¹⁵. To get where things can go wrong, we need to talk very briefly about the way rewriting works under the hood, although a precise explanation is far beyond the scope of this article. Essentially,

¹⁵There are some technical issues with rewriting of expressions involving bound variables. For example, rw [add_comm] fails to prove \forall a b : \mathbb{N} , a + b = b + a. Fortunately, there are some well-known workarounds, such as the use of $simp_rw$ instead of rw.

when trying to rewrite a=b, with a and b of type α , inside an expression e, Lean searches for some function φ , called the *motive*, such that e can be written as φ a. Rewriting a=b then simply amounts to the fact that a function maps equal arguments to equal values. The potential issue here is that the function φ has to be defined for all elements of type α , not just e and e.

This observation is enough to guarantee that rewriting a = b inside continuousFunctionalCalculus a f will essentially never work! Indeed, there are two separate reasons why continuousFunctionalCalculus x f would make no sense for a general element x: x need not be normal¹⁶, and f need not be defined (nor continuous) on the spectrum of x. Concretely, this means that essentially any attempt to do such a rewriting will lead to an error message of the form "Motive is not type correct...". We give two such examples in Listing 7. In the first one, we make sure that f is defined everywhere to isolate the failure due to non-normality of a general element. Conversely, in the second one, we work in a commutative C^* -algebra to demonstrate the failure due to f not being defined on the spectrum of a general element¹⁷.

```
Listing 7: Rewriting failures in the setting of Listing 1 Lean CFC/Snippets/Baseline.lean
70 example {a b : A} [IsStarNormal a] [IsStarNormal b] (hab : a = b) {f : C(\mathbb{C}, \mathbb{C})}
71    (H : continuousFunctionalCalculus a (f.restrict _) = 0) :
72    continuousFunctionalCalculus b (f.restrict _) = 0 := by
73    rw [hab] at H -- "motive is not type correct" because of 'IsStarNormal a'
74    sorry
```

```
Listing 8: Rewriting failures in the setting of Listing 1

LeanCFC/Snippets/Baseline.lean

[102 example {B : Type*} [CommCStarAlgebra B] {a b : B} (hab : a = b) {f : C(spectrum C a, C)}

[103 (H : continuousFunctionalCalculus a f = 0) :

[104 continuousFunctionalCalculus b (cast (by rw [hab]) f) = 0 := by

[105 rw [hab] at H -- "motive is not type correct" because of 'spectrum C a'

[106 sorry]
```

More generally, these issues tend to appear when a function takes arguments whose type depends on a former argument. As the approach in Listing 2 avoids types depending on a such as spectrum $\mathbb C$ a, IsStarNormal a, and StarAlgebra.elemental $\mathbb C$ a, rewriting works out of the box in the expected way, as demonstrated in Listing 9.

4.4. The composition property and its consequences. The issues mentioned in the previous section already strongly suggest to abandon the approach of Listing 1. Considering the composition property yields one more argument supporting the switch to Listing 2. Together with the fact that $f \mapsto f(a)$ is an isometric *-morphism, the composition property is probably the most crucial property for the applications of functional calculus, as well as the only way to link the functional calculi associated to distinct elements of the algebra. Thus, convenient use of this property and its consequences is of primary importance.

 $^{^{16}}$ And even if it were, the expression continuousFunctionalCalculus a f implicitly depends on a proof that a is normal, and that precise proof need not apply to x.

 $^{^{17}}$ The astute reader will notice that this already causes trouble in the statement of that example. Indeed, even though a = b, one cannot simply view f as having type C(spectrum $\mathbb C$ b, $\mathbb C$). Instead, one has to work with the function cast (by rw [hab]) f, which is essentially "f reinterpreted as having type C(spectrum $\mathbb C$ b, $\mathbb C$)".

In that regard, the approach of Listing 1 fails quite badly, as even stating the composition property is convoluted. Indeed, for the expression g(f(a)) to make sense, we need $f \in C(\sigma_{\mathbb{C}}(a), \mathbb{C})$ and $g \in C(\sigma_{\mathbb{C}}(f(a)), \mathbb{C}) = C(f(\sigma_{\mathbb{C}}(a)), \mathbb{C})$, and for such f and g one cannot formally write $g \circ f$. Instead, one has to consider the corestriction of f as a function $\hat{f}: C(\sigma_{\mathbb{C}}(a), \sigma_{\mathbb{C}}(f(a)))$, to be able to write $g(f(a)) = (g \circ \hat{f})(a)$. Notice in particular that this adds even more types depending on f and g, which means that the issues raised in the previous section could get even worse. On top of that, \hat{f} is only well-defined because of the spectral mapping theorem, which would thus need to be invoked repeatedly, and the co-restriction adds additional complexity to the expressions we are working with f. One can get a slightly cleaner statement by asking the user to provide the co-restricted avatar of \hat{f} , as demonstrated in Listing 10. This does give more flexibility for rewriting since one can delay the proof that the given \hat{f} indeed coincides with f, but this is still far from satisfying. We encourage the reader, without accessing Lean's interface, to try to parse the statement of the composition property in Listing 10; in particular, try to consider carefully with respect to which type the equality occurs, in which type the continuous functional calculus is applied, and why the relevant elements are normal. f

```
Listing 10: Statement of the composition property for the approach of Listing 1 LeanCFC/Snippets/Baseline.lean

108 theorem continuousFunctionalCalculus_map_comp {a : A} [IsStarNormal a]
109 {f : C(spectrum \mathbb{C} a, \mathbb{C})}
100 {f' : C(spectrum \mathbb{C} a, spectrum \mathbb{C} (continuousFunctionalCalculus a f))}
111 {g : C(spectrum \mathbb{C} (continuousFunctionalCalculus a f), \mathbb{C})}
112 (H : \forall x, f x = f' x) :
113 continuousFunctionalCalculus a (g.comp f') =
114 continuousFunctionalCalculus (continuousFunctionalCalculus a f) g := by
115 sorry
```

By contrast, stating and using the composition property in the approach of Listing 2 is as simple as it could be: since all functions are defined on the whole complex plane, one can compose them without extra work. Listing 11 contains this exact statement of the composition property — we omit the proof for now, as it will be discussed in Section 6 — and illustrates how to use it to show that $f(-a) = (x \mapsto f(-x))(a)$. We will see in Section 9 how to make this even more convenient.

```
Listing 11: Statement and use of the composition property for the approach of Listing 2
   LeanCFC/Snippets/Intermediate.lean
82 lemma cfc_comp {f g : \mathbb{C} \to \mathbb{C}} {a : A} [ha : IsStarNormal a] 83 (hg : ContinuousOn g (f '' spectrum \mathbb{C} a)) (hf : ContinuousOn f (spectrum \mathbb{C} a)) :
        cfc (g \circ f) a = cfc g (cfc f a) := by
84
85
86
87 lemma cfc_comp_neg \{f : \mathbb{C} \to \mathbb{C}\}\ \{a : A\}\ [ha : IsStarNormal a]
        (hf : ContinuousOn f (- spectrum ℂ a)) :
88
        cfc f (-a) = cfc (fun x \mapsto f (-x)) a := by
89
     rw [← Set.image_neg_eq_neg] at hf
90
     change ContinuousOn f ((-id) '' spectrum \mathbb C a) at hf
91
     rw [← cfc_id a, ← cfc_neg continuousOn_id, ← cfc_comp hf (by exact continuousOn_neg),
92
           cfc_id]
93
     rfl.
94
```

¹⁸This only gets worse with more iterations of composition, where one may need to use facts such as $h \circ \hat{g} \circ \hat{f} = h \circ g \circ \hat{f}$.

¹⁹Spoilers: The left-hand side is straightforward, a is normal by assumption, and the result lies in the subalgebra StarAlgebra.elemental $\mathbb C$ a. For the right-hand side, continuousFunctionalCalculus a f is an element of StarAlgebra.elemental $\mathbb C$ a, and the element is normal because this algebra is commutative. Then the outer continuousFunctionalCalculus application (with the function $\mathbf g$) is applied to this element yielding an element of the sub-subalgebra generated by f(a) inside the subagebra generated by a. Then the right-hand side is coerced to the subalgebra generated by a, so the equality takes place in StarAlgebra.elemental $\mathbb C$ a.

5. Using classes to parameterize the interface

The function cfc from Listing 2 in the previous section addresses some of the design constraints mentioned in Section 3, especially those pertaining to usability. However, as yet we have not addressed the issues related to generality also mentioned therein. In this section, we address how to do this by providing a class-based interface to the continuous functional calculus, still in the setting of *unital* algebras. Essentially, this takes the step of implementing Listing 2 from the isomorphism in Listing 1 and asks what features of Listing 1 are necessary to make this work. As such the discussion in this section is largely orthogonal to that of Section 4.

5.1. Alternate scalar rings. Listing 2 hard-codes the scalar ring \mathbb{C} , which means that utilizing functions $f: \mathbb{R} \to \mathbb{R}$ when a is selfadjoint is burdensome. Even trivial considerations would become onerous if we were to use it as a general framework. For example, if $f, g: \mathbb{R} \to \mathbb{R}$, it's trivial that (f+g)(a) = f(a) + g(a) because the continuous functional calculus is a *-homomorphism, but with the previous setup, we would first have to turn these functions into functions $f', g': \mathbb{C} \to \mathbb{C}$. On paper, this is trivial: simply expand the codomain and recall that for selfadjoint elements the spectrum is contained in \mathbb{R} , so it doesn't matter how f', g' are defined elsewhere.

In Lean, we would need to let $f' := \text{fun } x : \mathbb{C} \mapsto (\uparrow(f \text{ x.re} : \mathbb{C}))$, and then even showing (f+g)'(a) = f'(a) + g'(a) would involve multiple steps, as one first needs to establish (f+g)' = f' + g' (this is not hard, but it is an extra step). The situation only gets worse when one considers composition of functions, as the function $x : \mathbb{C} \mapsto x.re$ is ill-behaved.

An inelegant and maintenance-heavy solution to this problem would be to simply redo everything in triplicate, once for each of the scalar rings \mathbb{C} , \mathbb{R} and $\mathbb{R}_{\geq 0}$. This would be hard to maintain, as whenever a given declaration is changed, its corresponding versions would also need to be updated. A much better approach is to use a class to abstract over the scalar ring, and then to provide instances for \mathbb{C} , \mathbb{R} and $\mathbb{R}_{\geq 0}$. A class is a construct in Lean which provides a common interface to each instance of the class. For example, Add is a class in Lean providing access to the + notation; an instance Add α on the type α declares which binary function on α should be used when Lean encounters a + b for $a b : \alpha$.

In our case, we will be providing a class ContinuousFunctionalCalculus which provides the necessary information to construct the definition cfc over a given scalar ring R. A naive attempt to implement this is shown in Listing 12. Instances of this class would then be provided for normal, selfadjoint or nonnegative elements of the algebra²⁰, when R is \mathbb{C} , \mathbb{R} or $\mathbb{R}_{>0}$, respectively.

The reader should note that we are *not* undoing all the insights learned from the previous section; our goal is instead to provide the necessary mathematical context from which we can derive cfc generalized to other scalar rings. Unfortunately, Listing 12 does not accomplish this, and the issues are legion. Most importantly, in the case of $\mathbb{R}_{\geq 0}$, Listing 12 is not mathematically correct as $C(\sigma_{\mathbb{R}_{\geq 0}}(a), \mathbb{R}_{\geq 0})$ is not isomorphic to the closed *- $\mathbb{R}_{\geq 0}$ -algebra generated by a, but rather to the nonnegative elements in the *- \mathbb{R} -algebra generated by a.

The solution here is to no longer require that this is a *-isomorphism, and instead we require that it is a *-homomorphism into \mathcal{A} , but it requires slightly more finesse. In order to recover some of the nice properties of the continuous functional calculus, we need to somehow "remember" properties of the *-isomorphism which we can no longer reference directly. In the next attempt in Listing 13, we do this by means of the closedEmbedding and map_spectrum conditions. The former guarantees that the function is continuous, the

 $^{^{20}}$ The authors are aware that this wouldn't actually work as described (as IsSelfAdjoint and fun a \rightarrow 0 \leq a are not themselves classes), but we will explain this shortly.

range is closed, and the topology on the continuous functions $C(\sigma_R(a), R)$ coincides with the pullback of the topology on \mathcal{A} through the *-homomorphism toStarAlgEquiv. The latter is useful because the *spectral permanence* property is not automatic²¹ over other scalar rings. When R is either \mathbb{C} or \mathbb{R} , by the Stone–Weierstrass theorem, one can recover the *-isomorphism in Listing 12 from this closed embedding, which shows that we are not relinquishing any provability when $R := \mathbb{C}$.

```
Listing 13: Version 2: Avoiding the isomorphism
   LeanCFC/Snippets/Class.lean
28 class ContinuousFunctionalCalculus (R : Type*) {A : Type*} [CStarAlgebra A]
       [CommSemiring R] [StarRing R] [MetricSpace R] [TopologicalSemiring R]
29
       [ContinuousStar R] [Algebra R A] (a : A) where
30
     /-- The \star-homomorphism underlying the continuous functional calculus for 'a : A'. -/
31
    toStarAlgHom : C(spectrum R a, R) →*a[R] A
32
     /-- The ★-homomorphism sends the identity function on `spectrum R a` to `a`. -/
33
    map_id : toStarAlgHom (.restrict (spectrum R a) (.id R)) = a
34
    /-- The spectrum of the image of any function under the $*$-homomorphism is just
35
    the range of that function. -/
36
    map_spectrum (f : C(spectrum R a, R)) : spectrum R (toStarAlgHom f) = Set.range f
37
    /-- The ★-homomorphism is a closed embedding. -/
38
    closedEmbedding : IsClosedEmbedding toStarAlgHom
```

It turns out that there are still several issues with the class as given in Listing 13. The first is that a: A is an argument to the class, which means that definitions which depend on it (e.g., cfc) would be mired in rewriting, as discussed earlier in Section 4. Another issue is that Lean would not be able to find instances of ContinuousFunctionalCalculus, as they would depend on proofs that the element a satisfies the relevant predicate, and since the type does not depend on this predicate, nor is this predicate itself a class, Lean would not be able to infer this information during type class synthesis.

The solution to both of these problems is to reframe our perspective. Instead of thinking of a continuous functional calculus as a property of an element $a \in \mathcal{A}$, we should instead think of it as a property of the algebra \mathcal{A} itself which only provides information for elements satisfying a certain predicate. Note that generalizing to other scalar rings comes with the additional requirement that we must also generalize the predicate on elements of the algebra that are guaranteed to have a continuous functional calculus. In particular, when the scalar ring is \mathbb{C} , \mathbb{R} or $\mathbb{R}_{\geq 0}$ (the only ones we care about), the corresponding predicate on $a \in \mathcal{A}$ is that a is normal, selfadjoint, or nonnegative, respectively. This predicate depends only on the scalar ring²², and not the algebra itself. We therefore arrive at a workable version of the continuous functional calculus class, as shown in Listing 14.

```
Listing 14: Version 3: Bundling the predicate

LeanCFC/Snippets/Class.lean

46 class ContinuousFunctionalCalculus (R: Type*) {A: Type*} (p: outParam (A → Type*))

47  [CStarAlgebra A] [CommSemiring R] [StarRing R] [MetricSpace R]

48  [TopologicalSemiring R] [ContinuousStar R] [Algebra R A] where

49  toStarAlgHom {a} (ha: p a): C(spectrum R a, R) →*a[R] A

50  map_id {a} (ha: p a): toStarAlgHom ha (.restrict (spectrum R a) (.id R)) = a

51  map_spectrum {a} (ha: p a) (f: C(spectrum R a, R)):

52  spectrum R (toStarAlgHom ha f) = Set.range f
```

 $^{^{21}}$ In fact, when we generalize the type class assumptions in in the next subsection, this condition will be necessary because, since we no longer require $\mathcal A$ to be a C^* -algebra, so spectral permanence is no longer guaranteed. The most we would be able to show is that spectrum R f_sub = Set.range f where f_sub is the term of the range of toStarAlgHom a as a *-subalgebra (this is a subtype of A) whose coercion to A is toStarAlgHom a f. But without spectral permanence, the spectrum of an element in a subalgebra depends on the ambient algebra. That is, an element might be non-invertible in the subalgebra, but invertible in the full algebra; this is impossible in C^* -algebras.

²²Because it doesn't change, we mark it as an OutParam in the class definition, which means that Lean will automatically infer the value of this parameter when it is needed; this avoids the need to provide it explicitly every time we use the continuous functional calculus, which would be cumbersome.

```
closedEmbedding {a} (ha : p a) : IsClosedEmbedding (toStarAlgHom ha)
predicate_preserving {a} (ha : p a) (f : C(spectrum R a, R)) : p (toStarAlgHom ha f)
```

Note that there is now an additional field: predicate_preserving. This is valuable in order to establish the composition property without extraneous hypotheses.

5.2. Abstracting the C^* -algebra requirement. With the version of the continuous functional calculus class as given in Listing 14 we still cannot create an instance of this class for Matrix n n R, or even Matrix n n C. This is because the class requires that the algebra \mathcal{A} be a C^* -algebra. The type Matrix n n C does not have a (globally available) C^* -algebra instance in Mathlib because, as discussed in the introduction, Mathlib prefers not to choose a norm (or even a metric structure) on matrices. Likewise, Matrix n n R can never have a C^* -algebra instance because it is not an algebra over C. Fortunately, this is easily remedied: we simply remove the requirement that \mathcal{A} is a C^* -algebra in the definition of the class, and replace it with much weaker requirements. Namely, we only require that \mathcal{A} is a topological *-R-algebra^23. The changes to the type class assumptions are shown in Listing 15, as the structure fields coincide with those from Listing 14.

```
Listing 15: Version 4: Removing the C*-algebra constraint

LeanCFC/Snippets/Class.lean

61 class ContinuousFunctionalCalculus (R: Type*) {A: Type*} (p: outParam (A → Type*))

62 [CommSemiring R] [StarRing R] [MetricSpace R] [TopologicalSemiring R]

63 [ContinuousStar R] [Ring A] [StarRing A] [TopologicalSpace A] [Algebra R A] where
```

5.3. **Data and propositions.** The final problem that remains with the class in Listing 15 is that it contains data (i.e., a field which is a **Type** instead of a **Prop**) in the form of toStarAlgHom. The issue is that, for type classes to work effectively, the data in instances should be unique up to definitional, or judgmental, equality. There is no problem when all the fields of a class are **Prop**-valued because in Lean any two proofs of a proposition are definitionally equal by fiat, which is called proof irrelevance²⁴.

Mathematicians generally don't distinguish between definitional equality and *propositional* equality (i.e., provable equality), but in dependently typed proof assistants with an intensional type theory, like Lean, the distinction can be important. We won't delve into the details of this distinction here, but as a rough approximation: two terms are definitionally equal²⁵ if, after unfolding all definitions and without applying any theorems, the terms are the same. In Lean, you can check if $a b : \alpha$ are definitionally equal by writing example: a = b := rfl; then a and b are definitionally equal if and only if this compiles successfully. In contrast, two terms are propositionally equal if there is a proof that they are equal. So, for example, in Lean, for n : N, n + 0 = 0 is a definitional equality because addition is defined by recursion on the second variable, whereas 0 + n = 0 is a propositional equality because you can give a proof by induction, but it is not a definitional equality.

In order for Listing 15 to work effectively as a class, we would need to ensure that the toStarAlgHom field is unique up to definitional equality for any two instances we provide to Lean. This is actually problematic, as the following example with A := Matrix n n $\mathbb C$ shows. As mentioned previously, Mathlib does not equip matrices with a C^* -algebra structure, at least not globally²⁶. However, an instance of the class ContinuousFunctionalCalculus $\mathbb C$ IsStarNormal for Matrix n n $\mathbb C$ is still desirable, and this could be provided by means of diagonalization²⁷. The problem arises if one ever activates, even temporarily, a C^* -algebra structure on Matrix n n $\mathbb C$, as then there would be two instances of the continuous functional calculus available (one via abstract C^* -algebra theory and the Gelfand transform, and one via diagonalization), and their underlying star homomorphisms would not be definitionally equal.

 $^{^{23}}$ In fact, the requirements are even weaker: we only need \mathcal{A} to be a *-R-algebra which is also a topological space, without any compatibility requirements on those structures

²⁴This is also called *Uniqueness of Identity Proofs (UIP)*.

²⁵At the *default* transparency setting.

²⁶i.e., one that this always available, as opposed to a *scoped* instance which is only activated within a specified namespace.

 $^{^{27}}$ To be a bit more specific, if a: Matrix n n $\mathbb C$ is normal, then there is a unitary matrix u: Matrix n n $\mathbb C$ and a diagonal matrix d: Matrix n n $\mathbb C$ such that $a = u^H * d * u$, where u^H denotes the hermitian conjugate. Then the map fun f: $\mathbb C \to \mathbb C \mapsto u^H * (d.map f) * u$ is the desired star homomorphism underlying the continuous functional calculus.

Our solution relies on the following observation: the continuous functional calculus is *unique* due to the Stone–Weierstrass theorem (at least when R is \mathbb{C} or \mathbb{R} ; $\mathbb{R}_{\geq 0}$ is more finicky as we discuss in Section 6). Therefore, instead of providing the star algebra homomorphisms as data in the class, we instead only require that there exist star homomorphisms with the specified properties, thereby turning ContinuousFunctionalCalculus into a **Prop**-valued class and avoiding the problem of definitional equality entirely. This leads us to Listing 16, which is the version of ContinuousFunctionalCalculus present in Mathlib.

```
Listing 16: Version 5: Omitting data
   LeanCFC/Snippets/Class.lean
76 class ContinuousFunctionalCalculus (R : Type*) {A : Type*} (p : outParam (A → Prop))
        [CommSemiring R] [StarRing R] [MetricSpace R] [TopologicalSemiring R]
        ContinuousStar R] [Ring A] [StarRing A] [TopologicalSpace A]
78
        [Algebra R A] : Prop where
79
80
     predicate_zero : p 0
     [compactSpace_spectrum (a : A) : CompactSpace (spectrum R a)]
81
     spectrum_nonempty [Nontrivial A] (a : A) (ha : p a) : (spectrum R a).Nonempty
     exists_cfc_of_predicate : \forall a, p a \rightarrow \exists \varphi : C(spectrum R a, R) \rightarrow \star_a[R] A,
83
       IsClosedEmbedding \varphi \land \varphi ((ContinuousMap.id R).restrict \triangleleft spectrum R a) = a \land
84
          (\forall f, spectrum R (\phi f) = Set.range f) \land \forall f, p (\phi f)
85
```

The first three fields of this class are unrelated to our prior discussion. In fact, these fields are not strictly necessary, but they allow us to avoid adding type class assumptions downstream when developing general theory pertaining to the continuous functional calculus and are therefore convenient. In our first iteration that was merged to Mathlib, they were not present, and we will not mention them further. It is the last field exists_cfc_of_predicate which contains the key statement, now bundled into a singled proposition.

Given an instance of ContinuousFunctionalCalculus R p, we extract the star homomorphism ϕ appearing in Listing 16 into its own definiton:

```
Listing 17: Extracting the star homomorphism from the class

LeanCFC/Snippets/Class.lean

91 noncomputable def cfcHom {a : A} (ha : p a) : C(spectrum R a, R) →★a[R] A :=
92 (ContinuousFunctionalCalculus.exists_cfc_of_predicate a ha).choose
```

We can also define cfc analogously to Listing 2.

```
Listing 18: Defining cfc using the class

LeanCFC/Snippets/Class.lean

94 open scoped Classical in
95 noncomputable irreducible_def cfc (f : R → R) (a : A) : A :=
96 if h : p a ∧ ContinuousOn f (spectrum R a)
97 then cfcHom h.1 ⟨_, h.2.restrict⟩
98 else 0
```

6. Uniqueness

While the fact that the continuous functional calculus is a *-homomorphism is important, its most salient feature is the composition property. With the final version of the continuous functional calculus in hand, we can now state this property²⁸ concisely in Lean:

 $^{^{28}\}mathrm{This}$ is the same as the version mentioned in Listing 11, but now parametrized over R and p.

In the literature, this property is sometimes not stated at all ([Lin01, Corollary 1.3.6] or [KR97, Theorem 4.1.3 for selfadjoint elements], stated without proof [Tak10, p. 19], proven by polynomials and a direct appeal to the Stone–Weierstrass theorem [Dav96, Corollary I.3.3], or by appeal to the uniqueness of the continuous functional calculus [KR97, Theorems 4.4.5 and 4.4.8 for normal elements] or [Bou19, Proposition I.6.7 and Corollary I.6.2]).

We sketch the latter two approaches below.

Proof via polynomials. Let $a \in \mathcal{A}$ be a normal element in a C^* -algebra, and let $f: \sigma_{\mathbb{C}}(a) \to \mathbb{C}$ be a continuous function. Denote by f(a) the element of \mathcal{A} obtained by applying the continuous functional calculus for a to f. Then f(a) is normal and $\sigma_{\mathbb{C}}(f(a)) = f(\sigma_{\mathbb{C}}(a)) = \operatorname{range}(f)$ (using spectral permanence and the fact that the continuous functional calculus is a *-isomorphism onto the C^* -subalgebra generated by a). For any polynomial p in the variables p and p and p we have p and p the Stone-Weierstrass theorem, such polynomials are dense in the p algebra p and p are continuous, we obtain by taking limits p and p and p are continuous p and p are continuous, we obtain by taking limits p and p are p and p are continuous p and p are continuous, we obtain by taking limits p and p are p and p are continuous p and p are continuous, we obtain by taking limits p and p are p and p are continuous p and p are p and p and p are p are p and p are p are p and p are p are p and p are p and p are p are p are p and p are p and p are p are p and p are p are p and p are p and p are p are p and p are p are p are p and p are p are p and p are p are p and p are p are p are p and p are p are p are p and p are p and p are p are p are p are p are p are

Proof via uniqueness. Let $a \in \mathcal{A}$ be a normal element in a C^* -algebra. Note that if $\phi: C(\sigma_{\mathbb{C}}(a), \mathbb{C}) \to \mathcal{A}$ is any *-homomorphism such that $\phi(id) = a$, then $\phi(f) = f(a)$ for any continuous $f: \sigma_{\mathbb{C}}(a) \to \mathbb{C}$. Indeed, if for any polynomial p in the variables z and \bar{z} we have $\phi(p(z)) = p(\phi(id)) = p(a)$ Then by the Stone–Weierstrass theorem and continuity of ϕ , this extends to all continuous functions. Therefore the continuous functional calculus is the *unique* continuous *-homomorphism from $C(\sigma_{\mathbb{C}}(a), \mathbb{C})$ to \mathcal{A} sending the identity function to a.

Now suppose that $f: \sigma_{\mathbb{C}}(a) \to \mathbb{C}$ is continuous. There is the continuous functional calculus for the normal element f(a), which sends a continuous function $g: \sigma_{\mathbb{C}}(f(a)) \to \mathbb{C}$ to g(f(a)). However, for such g, notice that $g \circ f$ is a continuous function on $\sigma_{\mathbb{C}}(a)$, and so $(g \circ f)(a)$ is also meaningful, but is a priori different. Note that the mapping $g \mapsto (g \circ f)(a)$ is a continuous *-homomorphism from $C(\sigma_{\mathbb{C}}(f(a)), \mathbb{C})$ to \mathcal{A} sending the identity function (on $\sigma_{\mathbb{C}}(f(a))$) to f(a), and so by uniqueness, we must have $(g \circ f)(a) = g(f(a))$. \square

While both proofs appear in the literature, arguments akin to the former are ubiquitous throughout. For example, even though Kadison–Ringrose [KR97, Theorem 4.4.8] uses the uniqueness proof for the composition property, in [KR97, Propostion 4.2.3] the authors use the polynomial argument to show the positive and negative parts of a selfadjoint are unique, where a uniqueness argument would have worked just as well. Indeed, [KR97, Propostion 4.2.3] is implemented in Mathlib using uniqueness (CFC.posPart_negPart_unique²⁹).

Although the polynomial argument occurs frequently in the literature, we instead adopt the uniqueness approach in Mathlib for two primary reasons. The first is practical: limiting arguments by appeal to handwaving are acceptable on paper, but formalizing them can be cumbersome; uniqueness is simply easier. Additionally, we also need Listing 19 to hold when $R := \mathbb{R}_{\geq 0}$ in order for the continuous functional calculus to be useful, and unfortunately the Stone–Weierstrass theorem does not hold in this case³⁰. The astute reader will notice that the uniqueness argument *also* makes use of the Stone–Weierstrass theorem, but it

²⁹A copy of this proof is included in a code repository associated to this paper.

turns out that nevertheless uniqueness (and hence the composition property) still holds. So, we define the following class in Mathlib:

Note that whenever \mathcal{A} is a topological \mathbb{K} -algebra (with $\mathbb{K} := \mathbb{R}$ or $\mathbb{K} := \mathbb{C}$), then the Stone–Weierstrass theorem allows us to create an instance of this class. However, we can also provide an instance of this class for $R := \mathbb{R}_{\geq 0}$ provided that \mathcal{A} is a topological \mathbb{R} -algebra. The idea is, for $s \subseteq \mathbb{R}_{\geq 0}$ compact, to take a star algebra homomorphism $\phi : C(s, \mathbb{R}_{\geq 0}) \to \mathcal{A}$ (over $\mathbb{R}_{\geq 0}$), and create a star algebra homomorphism $\hat{\phi} : C(s, \mathbb{R}) \to \mathcal{A}$ (over \mathbb{R}) by defining $\hat{\phi}(f) := \phi(f_+) - \phi(f_-)$. Then the uniqueness for \mathbb{R} implies uniqueness for $\mathbb{R}_{\geq 0}$. So, making uniqueness of the continuous functional calculus into a *class* allows us unify the framework for \mathbb{C} , \mathbb{R} and $\mathbb{R}_{\geq 0}$.

One important question to consider is: why is it a separate class instead of bundled into the continuous functional calculus class itself? For starters, this class simply has instances in much greater generality (i.e., there is no need for \mathcal{A} to be a C^* -algebra at all). It is also dictated in part by the way we chose to obtain instances of continuous functional calculi for $\mathbb{R}_{\geq 0}$ and \mathbb{R} from those of \mathbb{R} and \mathbb{C} , respectively. We create a generic lemma for Continuous Functional Calculus that allows us to pass from a calculus over a scalar ring R with predicate p to a scalar subring S with predicate q, provided that any $a \in \mathcal{A}$ satisfies q if and only if it satisfies p and the R-spectrum of p is contained in p if we were to include the uniqueness criterion in the main class, then we would be forced to prove it when we establish this generic restriction lemma; this would be burdensome or impossible in that generality.

Another more important reason: we don't want uniqueness to hold only for *-homomorphisms into the algebra on which we have the continuous functional calculus, but indeed into any suitable algebra. This is essential to prove, for example, that the continuous functional calculus commutes with *-homomorphisms between C^* -algebras.

7. The continuous functional calculus for non-unital algebras

Non-unital C^* -algebras play a prominent role in the subject, but of course, the spectrum does not make sense for non-unital algebras, as there is no notion of invertibility. In textbooks on the subject, the approach is as follows: given a non-unital C^* -algebra \mathcal{A} , construct the minimal unitization \mathcal{A}^{32} \mathcal{A}^{+1} . This is the unital C^* -algebra $\mathcal{A}^{+1} := \mathbb{C} \times \mathcal{A}$ where multiplication is defined by (z,a)(w,b) := (zw,zb+wa+ab) and the norm is defined by $\|(z,a)\| := \max\{|z|, \|x\mapsto zx+ax\|\}$. It is called the minimal unitization because, for any unital C^* -algebra \mathcal{B} and any *-homomorphism $\phi: \mathcal{A} \to \mathcal{B}$, there is a unique unital *-homomorphism $\hat{\phi}: \mathcal{A}^{+1} \to \mathcal{B}$ extending ϕ .

At this point, textbook approaches diverge on their treatment, although all of these approaches are ultimately equivalent. We list a few of these approaches here. Kadison and Ringrose [KR97], for example, never touch non-unital C^* -algebras, as their focus primarily drifts to von Neumann algebras, which are always unital.

Huaxin Lin [Lin01] (even Pedersen [Ped79] and Blackadar [Bla06]) takes the approach of defining the spectrum of $a \in \mathcal{A}$ as the spectrum³³ of $(0, a) \in \mathcal{A}^{+1}$, but only when \mathcal{A} is actually non-unital (as opposed

³⁰More precisely, not every nonnegative continuous function is a uniform limit of polynomials with nonnegative coefficients. Indeed, such limits must be monotone functions on the nonnegative real numbers.

 $^{^{31}}$ As an example, $a \in \mathcal{A}$ is selfadjoint if and only if it is normal and its spectrum is contained in \mathbb{R} . Therefore we obtain a continuous functional calculus over \mathbb{R} for selfadjoint elements from the one over \mathbb{C} for normal elements.

³²There are actually two such unitizations which are widely used: one which, if the algebra is already unital, returns the algebra itself, and the other which always add a new identity element. It is the latter which we consider here.

to non-necessarily-unital). When \mathcal{A} is unital, the spectrum has the usual meaning. This approach is also employed by Davdison [Dav96] and Fillmore [Fil96], but they never explicitly define the spectrum for non-unital algebras, and instead freely pass to the unitization whenever it is convenient. In these approaches, the continuous functional calculus is a *-homomorphism from $C_0(\sigma_{\mathbb{C}}(a) \setminus \{0\}, \mathbb{C})$ to \mathcal{A} . Here, C_0 denotes the non-unital algebra of continuous functions vanishing at infinity, i.e., $f \in C_0(\sigma_{\mathbb{C}}(a) \setminus \{0\}, \mathbb{C})$ if, for every $\epsilon > 0$, $\{z \mid |f(z)| \ge \epsilon\}$ is compact.

Bourbaki [Bou19] and Dixmier [Dix69] take a different approach by defining two notions of spectrum: the usual spectrum in unital algebras, and another one for non-unital (i.e., not-necessarily-unital) algebras which is the spectrum in the unitization. The latter of these always contains zero. Bourbaki and Dixmier provide alternative notation for these two notions of spectrum, but Bourbaki at least still refers to the latter as "spectre de x relativement à \mathcal{A} ". In this variation, the continuous functional calculus is a *-homomorphism from the ideal of functions $f \in C(\sigma_{\mathbb{C}}((0,a)),\mathbb{C})$ for which f(0) = 0 to \mathcal{A} . Takesaki [Tak10, p. 7] utilizes the same technique, but refers to the spectrum in the unitization as the quasispectrum.

From the perspective of formalization, all of these textbook approaches have the downside that they depend upon the construction of a another type — the unitization (i.e., they are *extrinsic* formulations). In the case of [Lin01, Ped79, Bla06], the definition is even more problematic as it requires a case split on whether the algebra is unital or *actually* non-unital (as opposed to not-necessarily-unital). Our preferred approach is closest to Takesaki's in [Tak10], but we opt for an *intrinsic* definition of the quasispectrum for non-unital algebras.

Given an algebra \mathcal{A} over a (semi)field R, an element $r \in R$ is in the quasispectrum of $a \in \mathcal{A}$ if either r = 0 or $x := -(r^{-1}a)$ is not quasiregular (i.e., there is no $y \in \mathcal{A}$ such that x + y + xy = 0 = y + x + yx). We note that x is quasiregular in \mathcal{A} if and only if $(1,x) = r^{-1} \cdot (r, -a)$ is invertible in \mathcal{A}^{+1} ; consequently the quasispectrum of a in \mathcal{A} coincides with the spectrum of (0,a) in \mathcal{A}^{+1} (regardless of whether \mathcal{A} is unital or non-unital). It therefore coincides with the quasispectrum of a in the sense of [Tak10, p. 7]. We generally use the notation $\sigma_{n,R}(a)$ (or σ_n R a in Lean) for the quasispectrum of $a \in \mathcal{A}$ in the non-unital R-algebra \mathcal{A} .

This leads us to a definition of the continuous functional calculus for non-unital algebras which closely resembles the unital one. In the below, $C(\sigma_n \ R \ a, \ R)_0$ denotes the collection of continuous functions which vanish at zero.

```
Listing 22: Non-unital continuous functional calculus
   LeanCFC/Snippets/NonUnital.lean
17 class NonUnitalContinuousFunctionalCalculus (R : Type*) {A : Type*} (p : outParam (A → Prop))
        [CommSemiring R] [Nontrivial R] [StarRing R] [MetricSpace R] [TopologicalSemiring R]
       [ContinuousStar R] [NonUnitalRing A] [StarRing A] [TopologicalSpace A] [Module R A]
19
       [IsScalarTower R A A] [SMulCommClass R A A] : Prop where
20
21
     predicate_zero : p 0
     [compactSpace_quasispectrum : \forall a : A, CompactSpace (\sigma_n R a)]
     exists_cfc_of_predicate : \forall a, p a \rightarrow \exists \phi : C(\sigma_n \ R \ a, \ R)_\theta \rightarrow \star_{na}[R] A,
       24
         (\forall f, \sigma_n R (\varphi f) = Set.range f) \land \forall f, p (\varphi f)
25
```

³³Warning: if \mathcal{A} is unital, and one performs the same construction, then for $a \in \mathcal{A}$, the spectrum of $(0, a) \in \mathcal{A}^{+1}$ is $\sigma_{\mathbb{C}}(a) \cup \{0\}$, so it important to ensure that the algebra is *actually* non-unital when applying this definition.

As in the unital case, we also provide a uniqueness class for the non-unital continuous functional calculus.

Likewise we define cfcnHom and cfcn.

8. Instantiating the continuous functional calculus

Up to this point, we have only discussed the *design* of the continuous functional calculus, but not how to actually *instantiate* it. Of course, Listing 1 leads to a straightforward way to obtain an instance of ContinuousFunctionalCalculus $\mathfrak C$ IsStarNormal for unital C^* -algebras. But there are several other instances we need to provide, which include versions over $\mathbb R$ and $\mathbb R_{\geq 0}$ for unital algebras, as well as non-unital versions of all these. Moreover, we need an instance of the non-unital continuous functional calculus whenever we have an instance of the unital one. This last instance is necessary in order to, for example, take constructions involving the continuous functional calculus that work in non-unital algebras and use them in the unital setting as well (e.g., square roots of positive operators, positive and negative parts of a selfadjoint operator).

8.1. Passing to scalar subrings. On paper, other than proving that selfadjoint elements have real spectrum, there is virtually nothing to be done to obtain the continuous functional calculus over \mathbb{R} for selfadjoint elements from the one over \mathbb{C} for normal elements. This is so much the case that it is often not even mentioned. For nonnegative elements, generally even less is said, as in textbooks nonnegative elements are often defined as selfadjoint elements with nonnegative spectrum.

In contrast, we definitely need this in Lean because \mathbb{R} and \mathbb{C} are separate types, and it is much nicer, where possible, to work with \mathbb{R} directly rather than its image in \mathbb{C} . But at the same time, this needs to be explicit. Because we have to do this four times (\mathbb{C} to \mathbb{R} , \mathbb{R} to $\mathbb{R}_{\geq 0}$, for both unital and non-unital functional calculi), we developed a suitable interface for indicating that the spectrum of an element with respect to a given scalar ring can be reinterpreted as the spectrum relative to a scalar subring. We introduce the structure QuasispectrumRestricts, described in Listing 26, for this purpose. In fact, we use the exact same structure (up to definitional equality) for the *spectrum* in unital algebras as well. This is because, over a (semi)field,

the quasispectrum is contained in a scalar subfield if and only if the spectrum is contained in that subfield, since they differ at most by the presence of zero.

```
Listing 26: Restriction of the (quasi)spectrum
  LeanCFC/Snippets/Instances.lean
5 /-- Given an element 'a : A' of an 'S'-algebra, where 'S' is itself an 'R'-algebra, we say
6 that the spectrum of 'a' restricts via a function 'f : S \rightarrow R' if 'f' is a left inverse of
  'algebraMap R S', and 'f' is a right inverse of 'algebraMap R S' on 'spectrum S a'.
9 For example, when 'f = Complex.re' (so 'S := \ell' and 'R := R'), 'SpectrumRestricts a f'
10 means that the 'C'-spectrum of 'a' is contained within 'R'. This arises naturally when
11 'a' is selfadjoint and 'A' is a C∗-algebra. -/
12 structure QuasispectrumRestricts {R S A : Type*} [CommSemiring R] [CommSemiring S]
      [NonUnitalRing A] [Module R A] [Module S A] [Algebra R S] (a : A) (f : S \rightarrow R) : Prop where
    /-- `f` is a right inverse of `algebraMap R S` when restricted to `quasispectrum S a`. -/
14
    rightInvOn : (quasispectrum S a).RightInvOn f (algebraMap R S)
15
    /-- 'f' is a left inverse of 'algebraMap R S'. -/
    left_inv : Function.LeftInverse f (algebraMap R S)
```

We then prove a general theorem (Listing 27) that if we have a continuous functional calculus over a scalar ring S for an algebra \mathcal{A} with predicate q, and p is some other predicate which is equivalent to q and that spectrum restricts to a scalar subring R, then we can obtain a continuous functional calculus over R for \mathcal{A} with predicate p, under the assumption that the natural map from R to S is a uniform embedding. Of course, we can do the same for the non-unital continuous functional calculus as well. This general theorem can't be given as an *instance* because Lean would not be able to infer it in practice, but it allows us to provide very short proofs of the relevant instances.

```
Listing 27: Restricting the continuous functional calculus to scalar subrings
   LeanCFC/Snippets/Instances.lean
19 /-- Given a 'ContinuousFunctionalCalculus S q'. If we form the predicate 'p' for 'a : A'
20 characterized by: 'q a' and the spectrum of 'a' restricts to the scalar subring 'R' via
  f: C(S, R), then we can get a restricted functional calculus
  'ContinuousFunctionalCalculus R p'. -/
theorem SpectrumRestricts.cfc {R S A : Type*} {p q : A → Prop} [Semifield R] [StarRing R]
23
       [MetricSpace R] [TopologicalSemiring R] [ContinuousStar R] [Semifield S] [StarRing S]
24
        MetricSpace S] [TopologicalSemiring S] [ContinuousStar S] [Ring A] [StarRing A]
25
        ʿAlgebra S A] [Algebra R S] [Algebra R A] [IsScalarTower R S A] [StarModule R S]
26
27
       [ContinuousSMul R S] [TopologicalSpace A] [ContinuousFunctionalCalculus S q]
       [CompleteSpace R] (f : C(S, R)) (halg : IsUniformEmbedding ゕ(algebraMap R S))
       (h0 : p 0) (h : ∀ (a : A), p a ↔ q a ∧ SpectrumRestricts a <math>nf) :
29
       ContinuousFunctionalCalculus R p :=
30
```

- 8.2. The non-unital instance for unital algebras. Of course, if we have an instance of the continuous functional calculus for a unital algebra, then we can obtain an instance of the non-unital continuous functional calculus for the same algebra. This might seem so obvious as to be trivial, but there is some minor subtlety. Of course, given a star homomorphism $\phi: C(\sigma_R(a), R) \to \mathcal{A}$, we want to construct non-unital star homomorphism $\phi_0: C(\sigma_{n,R}(a), R)_0 \to \mathcal{A}$. This is not actually so hard to do: we simply chain together the maps $(\uparrow): C(\sigma_{n,R}(a), R)_0 \to C(\sigma_{n,R}(a), R)$ (this is just sending the function to itself meanwhile forgetting the fact that it maps zero to zero; in Lean it is represented as a coercion), $\psi: C(\sigma_{n,R}(a), R) \to C(\sigma_R(a), R)$ and ϕ , where ψ is given by precomposition with the inclusion map from $\sigma_R(a)$ to $\sigma_{n,R}(a)$. The minor challenge comes in proving that the resulting composition so-formed is a closed embedding (given that ϕ itself is too). Indeed, while ϕ and (\uparrow) are always closed embeddings, ψ in general is not if a is not invertible. Instead, the key is is that $\psi \circ (\uparrow)$ is always a closed embedding.
- 8.3. The non-unital instance for non-unital algebras. It is also necessary to construct an instance of the non-unital functional calculus (over \mathbb{C} , other scalar subrings are obtained from this via the generic framework above) for non-unital C^* -algebras. However, no matter how it is done (e.g., via the character

space or maximal ideal space), the unital version rears its head in the proof. This is not surprising, given the contravariant equivalence of categories between (non-unital) commutative C^* -algebras and (respectively, pointed) compact Hausdorff spaces. Given that we are not as interested in the isomorphism, but instead we primarily want an instance of NonUnitalContinuousFunctionalCalculus, we opt for an approach which differs from that in most textbooks on the subject.

Suppose that \mathcal{A} is a non-unital C^* -algebra. Then we can form the minimal unitization (over \mathbb{C}) \mathcal{A}^{+1} . Because this is a unital C^* -algebra, we have a continuous functional calculus on \mathcal{A}^{+1} over \mathbb{C} for normal elements. Then we can construct a non-unital star homomorphism $\phi: C(\sigma_{n,\mathbb{C}}(a),\mathbb{C})_0 \to \mathcal{A}^{+1}$ by chaining together the maps: $(\uparrow): C(\sigma_{n,\mathbb{C}}(a),\mathbb{C})_0 \to C(\sigma_{n,\mathbb{C}}(a),\mathbb{C})$ which is the aforementioned coercion; the star isomorphism from $C(\sigma_{n,\mathbb{C}}(a),\mathbb{C})$ to $C(\sigma_{\mathbb{C}}((0,a)),\mathbb{C})$ (where $(0,a) \in \mathcal{A}^{+1}$) arising from the set-equality $\sigma_{n,\mathbb{C}}(a) = \sigma_{\mathbb{C}}((0,a))$; and the (unital!) continuous functional calculus $C(\sigma_{\mathbb{C}}((0,a)),\mathbb{C}) \to \mathcal{A}^{+1}$.

This is not quite the map we want because it takes values in \mathcal{A}^{+1} , not \mathcal{A} . The last step is then to realize that, because $C(\sigma_{n,\mathbb{C}}(a),\mathbb{C})_0$ is generated (as a topological star algebra) by the identity function (this is Stone–Weierstrass), and the image of the identity is $(0,a) \in \mathcal{A}^{+1}$, then the range of this non-unital star homomorphism is actually contained in the image of \mathcal{A} in \mathcal{A}^{+1} . Since this inclusion is a closed embedding, we can pull back the map described above to get a closed embedding (and a non-unital star homomorphism) $C(\sigma_{n,\mathbb{C}}(a),\mathbb{C})_0 \to \mathcal{A}$.

9. AUTOPARAM AND AUTOMATION

In Section 4, we described in detail how important it was for us to unbundle the continuous functional calculus so that it is both itself a bare function, and also that it operates on bare functions. However, there is one significant downside of unbundling: we instead must pass around proofs that the element $a \in \mathcal{A}$ satisfies the predicate p, the function f is continuous on $\sigma_{\mathbb{C}}(a)$ and, in the non-unital case, f(0) = 0. This is cumbersome and a bit annoying, especially since oftentimes the proofs are available in the context, or easily derived from the current context, or unavailable but easily constructed.

Our solution to this conundrum is to use the autoParam feature of Lean. This allows the user, when writing a theorem, to specify a default tactic to use to attempt to generate a proof of one of the hypotheses. This tactic is tried when the user does not provide a proof for that hypothesis. As an example, consider the theorem:

The arguments hf, hf0 and ha each have an autoParam, and if, when this theorem is called, they are not provided, then the tactics cfc_cont_tac, cfc_zero_tac and cfc_tac are used to try to construct them. Currently, each of these tactics is just a wrapper around a small collection of tactics, but in the future, we can opt to make these more sophisticated, if it is useful. As an example, this can be used to show:

The underscores in the call to $cfc_n_map_quasispectrum$ are for the arguments to f := sqrt and a := b * b, which Lean can infer via unification in this case. The argument ha is constructed by the tactic cfc_tac , which is mostly a wrapper around the general purpose tactic aesop. In this case, IsSelfAdjoint.mul_self_nonneg³⁴ is the relevant lemma marked with the aesop attribute, which aesop can then combine with the hypothesis hb

 $^{^{34}}$ This is the lemma:

to show that b * b is nonnegative (this is necessary because we're using the functional calculus over R≥0 in this example). Likewise, the argument hf is constructed by the tactic cfc_cont_tac, which is a wrapper around fun_prop — a general purpose tactic for proving goals related to properties of functions (e.g., continuity, differentiability, measurability, etc.). Finally, the argument hf0 is constructed by the tactic cfc_zero_tac, which is also a wrapper around aesop; in this case it applies the simp lemma NNReal.sqrt_zero.³⁵

The net effect of this setup is that the user is able to freely apply lemmas related to the continuous function calculus, without providing proofs of the hypotheses, so long as the necessary proofs are sufficiently simple. We remark that unlike most other places in the library, the arguments f and a are intentionally left explicit, rather than implicit, despite the fact that they can be inferred from hf, hf0 and ha. This is exactly because the latter are autoParams, and since these are not provided by the user but rather autogenerated, sometimes Lean does not have enough information to infer f and a.

10. The isometric variation

Although we took great pains to ensure that the continuous functional calculus would be usable in contexts where a CStarAlgebra instance is not available, it is nevertheless the case that within the context of C^* -algebras, the fact that the continuous functional calculus is isometric is valuable and important. There are two ways this can be addressed. The first is to simply prove a theorem of the form $\|\mathsf{cfc} \ \mathsf{f} \ \mathsf{a}\| = \|\mathsf{f}\|$ under suitable conditions³⁶ on f and $\mathsf{a} : \mathsf{A}$, in the presence of a CStarAlgebra A instance. The second is to provide a separate class for the isometric continuous functional calculus, which simply extends the class specified in Listing 16 by requiring that the homomorphism therein is isometric.

The latter option may seem pointless, as it will only be applicable when A has a metric structure, which effectively means it only applies when there is already a CStarAlgebra instance. However, we have indeed opted for the latter option, and there were two considerations which led us to this decision. The first is that, because there is not one (unital) continuous functional calculus, but rather three (over \mathbb{C} , \mathbb{R} and $\mathbb{R}_{\geq 0}$), we would have to prove every theorem in triplicate; by using classes, we can have lemmas that apply to all three scalar rings simultaneously³⁷. The second consideration is that, in the future, we may develop the theory of real C^* -algebras, and in that case, having a separate class that can apply in both the complex and real settings will be vital.

In this way, we can obtain theorems that apply to a continuous functional calculus over either \mathbb{R} or \mathbb{C} . For instance, for every element x in the spectrum of a, the norm of f at x is bounded by ||f(a)||:

Because the continuous $\mathbb{R}_{\geq 0}$ functions don't have a norm structure in Mathlib, we sometimes state the corresponding theorems for the continuous functional calculus over $\mathbb{R}_{\geq 0}$ separately. The example below is the $\mathbb{R}_{\geq 0}$ version of the previous theorem.

```
lemma IsSelfAdjoint.mul_self_nonneg {R : Type*} [NonUnitalSemiring R] [PartialOrder R] [StarRing R] [StarOrderedRing R] {a : R} (ha : IsSelfAdjoint a) : 0 ≤ a * a :=
```

 $^{^{35}}$ This is the lemma: NNReal.sqrt_zero : sqrt 0 = 0.

 $^{^{36}\}mathrm{We}$ can't actually use the norm of f here since it is a bare function.

 $^{^{37}}$ Actually, this is not *quite* true, as $\mathbb{R}_{\geq 0}$ poses some unique challenges and somewhat often we need separate theorems for this scalar ring. Nevertheless, this approach does significantly reduce duplication between \mathbb{R} and \mathbb{C} .

```
theorem apply_le_nnnorm_cfc_nnreal {A : Type*} [NormedRing A] [StarRing A]

[NormedAlgebra R A] [PartialOrder A] [StarOrderedRing A]

[IsometricContinuousFunctionalCalculus R A IsSelfAdjoint] [NonnegSpectrumClass R A]

(f : NNReal → NNReal) (a : A) {x : NNReal} (hx : x ∈ spectrum NNReal a)

(hf : ContinuousOn f (spectrum NNReal a) := by cfc_cont_tac) (ha : 0 ≤ a := by cfc_tac) :

f x ≤ ||cfc f a|| + :=
```

11. Limitations and pain points

Overall, we feel that our implementation of the continuous functional calculus is quite successful. Nevertheless, there are still some rough edges that we would like to polish.

11.1. Lack of simp lemmas. Because of our choice to use bare functions everywhere, almost every single lemma (with the exception of cfc_zero and $cfc_{n-}zero$) has some hypotheses that must be satisfied which cannot be filled by unification. As a result, these lemmas are not suitable simp lemmas.

For instance, the lemma cfc_id states that cfc id a = a, but it requires a proof that a satisfies the predicate p pertaining to the scalar ring R appearing in id : $R \rightarrow R$. As outlined in Section 9, this argument is equipped with an autoParam which can be used to automatically construct this proof in many circumstances. One would hope that this would be sufficient to make cfc_id a simp lemma³⁸ which is only applied in contexts where the proof construction succeeds.

Unfortunately, due to technical limitations³⁹, this is not the case. That is, Lean needs the user to provide the explicit arguments for f and a in order to force Lean to generate the proofs via autoParam. This makes some proofs more tedious than is preferable, as one must generally provide all the rewrite steps explicitly, even the "obvious" ones.

11.2. The headache of $\mathbb{R}_{\geq 0}$. The use of $\mathbb{R}_{\geq 0}$ as a scalar ring is a double-edged sword. While it makes manipulation of nonnegative elements quite natural, it also introduces some complications. We often have to prove specialized versions of theorems for $\mathbb{R}_{\geq 0}$, and generic theorems don't apply. One example was given in Section 10, but we'll list a few more here. The lemma cfc_le_iff (and its non-unital counterpart cfc_le_iff) shown below are only valid when the scalar ring R is ring, not just a semiring. Consequently, we have to write a separate version (cfc_nnreal_le_iff) of this lemma for $\mathbb{R}_{\geq 0}$.

```
theorem cfc_le_iff {R : Type u} {A : Type*} {p : A → Prop} [OrderedCommRing R]

[StarRing R] [MetricSpace R] [TopologicalRing R] [ContinuousStar R]

[∀ (α : Type u) [TopologicalSpace α], StarOrderedRing C(α, R)]

[TopologicalSpace A] [Ring A] [StarRing A] [PartialOrder A]

[StarOrderedRing A] [Algebra R A] [ContinuousFunctionalCalculus R p]

[NonnegSpectrumClass R A] (f g : R → R) (a : A)

[Hf : ContinuousOn f (spectrum R a) := by cfc_cont_tac)

[Hg : ContinuousOn g (spectrum R a) := by cfc_cont_tac)

[Ha : p a := by cfc_tac) :

[Ha : cfc f a ≤ cfc g a ↔ ∀ x ∈ spectrum R a, f x ≤ g x :=
```

Likewise, the lemma cfc_sub (which simply states that cfc (f - g) a = cfc f a - cfc g a) also requires the scalars to be an actual ring. The problem is that $\mathbb{R}_{\geq 0}$ utilizes truncated subtraction, wherein x-y:=0 when $x\leq y$.

Another annoyance is related to the unitization, especially as regards the function spaces C(s,R) and $C(s,R)_0$. When $0 \in s$, and R is a topological ring (not just a topological semiring), then there is a natural ring isomorphism $\Phi: C(s,R) \to C(s,R)_0^{+1}$, where the unitization is taken over R. Here, $\Phi(f) := (f(0), f - f(0))$, and $\Phi^{-1}(a,g) := a + g$. However, this isomorphism clearly falters when R is only a semiring, these rings are not isomorphic in that case.

 $^{^{38}}$ Potentially, this would be placed in a separate simp set, so that these lemmas (with the associated proof search) are not tried on every simp invocation that matches the pattern.

 $^{^{39}}$ For more details, see https://github.com/leanprover/lean4/issues/3475

12. Future Work

12.1. A cfc manipulation tactic. By perusing the code artifact associated to this paper, a careful reader may notice that the proofs often involve rewrites (or simp calls) that rewrite in the reverse direction (i.e., with \leftarrow) of the lemma statement. From a mathematical perspective, the reason is clear: in order to show that two expressions involving the continuous functional calculus are equal, it suffices to write both as applications of the continuous functional calculus for some element $a \in \mathcal{A}$ to some functions $f, g : \mathbb{C} \to \mathbb{C}$ and then show that f, g agree on the spectrum of a. This last step is the lemma cfc_congr or cfc_n_congr that appears repeatedly. And of course, this is a natural thing to do because working with functions is simpler than working with elements in the C^* -algebra.

The reader may wonder why we state all these lemmas in the wrong direction. While this is something we could switch, and maybe we should do so at some point, there is a reason it ended up this way. Throughout Mathlib, when $\phi: \mathcal{A} \to \mathcal{B}$ is a morphism of semigroups (or anything stronger, such as a star homomorphism), the lemma map_mul applies, which states that $\phi(xy) = \phi(x)\phi(y)$ for $x,y \in \mathcal{A}$. Moreover, this lemma is a simp lemma, and so it is applied automatically whenever that tactic is called. Recalling that cfc itself (or more precisely, cfcHom) is a star homomorphism, we can see that cfc_mul is just a special case of map_mul.

To emphasize, lemmas in the library which apply to morphisms and are marked simp (e.g., map_mul) generally push the morphism to the *leaves* of the expression (viewed as tree). In contrast, when applying the analogous lemmas (such as cfc_mul) for the continuous functional calculus, we generally want to pull cfc to the *head* of the expression, so it's exactly the reverse of what we do normally. Of course, often we *really do* want the lemmas in the direction currently stated (e.g., cfc_id), and we would like it if they were simp lemmas (but only in this direction). At the same time, when moving cfc to the head of an expression, even cfc_id is used in the opposite direction.

This suggests that what we truly need is a tactic that can manipulate expressions involving the continuous functional calculus in an intelligent way. Such a tactic should be able to pull cfc to the head of an expression, or push it to the leaves as necessary. As it applies lemmas, it should collect goals that cannot be solved by the existing automation, and leave them for the user to prove. This would reduce the burden on the user to continually provide the entire list of rewrites, and, if written properly, would avoid the need for many targeted rewrites like the invocation $nth_rw 2 \in cfc_n_id R a$.

12.2. **Real** C^* -algebras. Throughout the development process, we have been careful to ensure that our implementation is as general as possible. In particular, we want this to be usable for real C^* -algebras when those eventually enter Mathlib. Note that unlike most areas of mathematics where the real theory may generally be developed in parallel, or even prior to, the complex theory, in the context of C^* -algebras, the complex theory must be developed first. Indeed, even the spectrum of an element in a real C^* -algebra is a subset of the complex plane (and not just a subset of the real line). One way to understand the reason for this is that the category of real C^* -algebras is equivalent to the category of complex C^* -algebras equipped with a conjugate-linear multiplicative involution (or equivalently, a linear antimultiplicative involution).

Our implementation of the continuous functional calculus over \mathbb{R} and $\mathbb{R}_{\geq 0}$ should work out-of-the-box for real C^* -algebras, once the relevant instances are supplied. On the other hand, the continuous functional calculus over \mathbb{C} (for normal elements) is markedly different in the case of real C^* -algebras. In particular, it is not a star homomorphism from $C(\sigma_{\mathbb{C}}(a),\mathbb{C})$ into \mathcal{A} , but the domain is rather the ideal of those continuous functions which commute with complex conjugation (i.e., $f(\overline{z}) = \overline{f(z)}$). Given that this is sufficiently different from all the other cases, our current attitude is that this should be handled via a bespoke interface, even if the proofs ultimately boil down to an appropriate application of the continuous functional calculus over \mathbb{C} on a certain complex C^* -algebra.

CODE ARTIFACT

Code associated with this paper is available at https://github.com/j-loreaux/LeanCFC/

Acknowledgements

The authors thank Frédéric Dupuis for many thoughtful discussions during the development of this project, and for much code review upon submission to Mathlib. More generally, we're indebted to the entire Mathlib and Lean communities, without whose efforts this work would not have been possible.

References

- [Baa22] Anne Baanen, Use and abuse of instance parameters in the lean mathematical library, 13th International Conference on Interactive Theorem Proving (ITP 2022), vol. 237, Schloss Dagstuhl Leibniz-Zentrum für Informatik, 2022, pp. 1–20 (English).
- [Bla06] Bruce Blackadar, Operator algebras. Theory of C*-algebras and von Neumann algebras, Encycl. Math. Sci., vol. 122, Berlin: Springer, 2006 (English).
- [Bou19] Nicolas Bourbaki, Théories spectrales, 2nd ed. ed., Springer International Publishing AG, Cham, 2019.
- [Dav96] Kenneth R. Davidson, C*-algebras by example, Fields Inst. Monogr., vol. 6, Providence, RI: AMS, American Mathematical Society, 1996 (English).
- [Del09] Ivo Dell'Ambrogio, Categories of c^* -algebras, 2009.
- [Dix69] Jacques Dixmier, Les C^* -algèbres et leurs représentations. 2e ed, Cahiers scientifiques. 29. Paris: Gauthier-Villars. XV, pp. 390, 1969 (French).
- [Fil96] Peter A. Fillmore, A user's guide to operator algebras, Canadian Mathematical Society Series of Monographs and Advanced Texts, New York, NY: Wiley, 1996 (English).
- [Gel41] Israel M. Gelfand, Normierte Ringe, Rec. Math. Moscou, n. Ser. 9 (1941), 3-24 (German).
- [KR97] Richard V. Kadison and John R. Ringrose, Fundamentals of the theory of operator algebras, Graduate Studies in Mathematics), no. volume 15, American Mathematical Society, Providence, R.I, 1997, Second printing, incorporating minor corrections, of the work originally published: New York: Academic Press, 1983. (Pure and applied mathematics (Academic Press); 100). - Includes bibliographical references and indexes. - Electronic reproduction; Providence, Rhode Island; American Mathematical Society; 2012. - Description based on print version record.
- [Lin01] Huaxin Lin, An introduction to the classification of amenable C*-algebras, Singapore: World Scientific, 2001 (English).
- [mC20] The mathlib Community, The lean mathematical library, Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, POPL '20, ACM, January 2020.
- [Neg71] Joan W Negrepontis, Duality in analysis from the point of view of triples, Journal of Algebra 19 (1971), no. 2, 228–253.
- [Ped79] Gert K. Pedersen, C*-algebras and their automorphism groups, Lond. Math. Soc. Monogr., vol. 14, Academic Press, London, 1979 (English).
- [Tak10] Masamichi Takesaki, Takesaki, 2nd printing of the 1st edition 1979 ed., vol. 1, Encyclopaedia of mathematical sciences, no. 124, Springer, Berlin, 2010.

Université Paris Cité, Sorbonne Université, CNRS, IMJ-PRG, F-75013 Paris, France

Southern Illinois University Edwardsville, Edwardsville, Illinois 62026, USA