How a Tree Remembers Its Cuts

The Connes–Kreimer–Foissy Hopf algebra of rooted trees, machine-checked in Lean 4

Carles Marín*
Independent researcher
karlesmarin@gmail.com

June 2026

Abstract

Take a rooted tree and cut it: each admissible cut splits it into a pruned forest and a trunk, and the sum over all cuts is a coproduct Δ. The remarkable fact—the fact that gives perturbative renormalization its algebra—is that these cuts are *coherent*: cutting twice in either order agrees ($(\Delta \otimes \mathrm{id})\Delta = (\mathrm{id} \otimes \Delta)\Delta$), so the trees, their disjoint-union product, this coproduct, a counit, and an antipode assemble into a *Hopf algebra*. This is the Connes–Kreimer algebra of renormalization and, read another way, the Butcher algebra of Runge–Kutta methods. This paper formalizes it in Lean 4 / Mathlib, all `sorry`-free: the planar (Foissy, noncommutative) algebra of rooted trees is registered as a genuine Mathlib `HopfAlgebra` instance. To the best of our knowledge, after a focused search of the Lean/Mathlib, Isabelle/HOL and Coq/Rocq libraries, no combinatorial renormalization Hopf algebra of rooted trees had previously been formalized in a proof assistant (Section 8). The build splits into four checkable claims. The coproduct is defined by the Grossman–Larson/B_+ recursion; coassociativity—about 80% of the work—is proved combinatorially up to permutation of cuts and then lifted to the Mathlib equality of linear maps, yielding the `Coalgebra` instance. The product is shown compatible (`Bialgebra`). The antipode is built by connected-graded well-founded recursion on the node count; its *left* axiom $\mu(S \otimes \mathrm{id})\Delta = \eta\varepsilon$ is near-definitional, and the *right* axiom is obtained not by re-running the recursion but by constructing a mirror antipode S' and proving $S = S'$ inside Mathlib's convolution monoid `WithConv`—a left inverse that admits a right inverse must coincide with it. I am exact about the boundary: this is the *planar* (noncommutative) version on the quotient-free `List`-based tree type, not yet the commutative Butcher/Connes–Kreimer quotient; the mathematics is classical throughout; and "first in a proof assistant" is a focused multi-prover search result, not a theorem. The contribution is the formalization, and a reusable certified core—trees, the cut coproduct, coassociativity, and a connected-graded antipode—on which B-series order conditions, the free pre-Lie algebra, and Birkhoff renormalization can later be built.

Reader's map. If you read only two things, read Theorem 1 (coassociativity, hence the `Coalgebra`) and Theorem 3 (the `HopfAlgebra`). The first says the cuts of a tree cohere; the second says the whole structure—product, coproduct, counit, antipode—closes into a Hopf algebra. Everything in between (the objects, Section 2; the coproduct, Section 3; the bialgebra, Section 5; the antipode, Section 6) builds toward them. The load-bearing formulas are framed; the honest limits (Section 8) are stated as plainly as the results.[1]

*A large language model was used as a coding assistant; every statement was independently verified by the Lean kernel, and all mathematics and claims are the author's responsibility.

[1] *Reproducibility.* The Lean source is the single self-contained file `ConnesKreimer.lean`, built against Lean

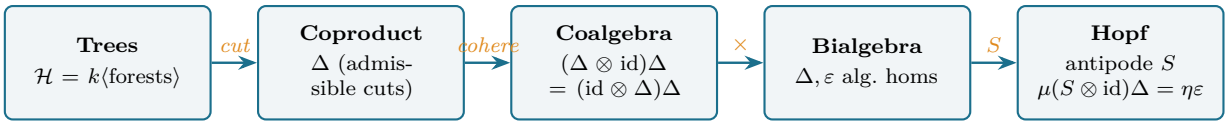


Figure 1: The spine of the paper. Cutting a rooted tree defines a coproduct (the Grossman–Larson/B_+ recursion); the cuts *cohere* (coassociativity), giving a coalgebra; the disjoint-union product is compatible (bialgebra); and a connected-graded antipode closes the structure into a Hopf algebra. Each arrow is a machine-checked theorem of this paper, and each box a Mathlib instance.

1 Introduction: three readings of one algebra

The combinatorics of subtracting divergences in perturbative quantum field theory, the order conditions of Runge–Kutta methods, and the free pre-Lie algebra are governed by one and the same Hopf algebra of rooted trees. This paper formalizes that Hopf algebra in Lean 4.

The objects are the simplest in combinatorics: *rooted trees*, which Cayley counted in the nineteenth century [3], and *forests* (disjoint unions of them). The structure on top of them has three histories that converge.

The first is *numerical analysis*. In the 1960s and 70s John Butcher organized the algebra of Runge–Kutta methods around rooted trees: each tree indexes an elementary differential of the vector field, and the condition that a method have order p is a finite set of equations indexed by trees up to size p [2, 8]. The *composition* of numerical flows is a group law on tree-indexed series—the Butcher group—and its underlying Hopf algebra is exactly the one below.

The second is *renormalization*. In 1998 Connes and Kreimer discovered that the combinatorics of subtracting divergences in perturbative quantum field theory is governed by a Hopf algebra of rooted trees: the coproduct enumerates the subdivergences (the admissible cuts), the antipode produces the counterterms, and the BPHZ forest formula is a Birkhoff decomposition in the corresponding pro-unipotent group [5, 9]. Brouder later observed that the Connes–Kreimer and Butcher algebras coincide [1]: renormalization and Runge–Kutta are two readings of one structure.

The third is *algebra proper*. Foissy gave a *noncommutative*, or *planar*, version on trees with ordered children [6], and Chapoton–Livernet identified the rooted-tree operad with the free pre-Lie algebra [4]; Grossman–Larson had earlier built a dual Hopf algebra directly from the grafting product [7]. Manchon's survey ties the strands together and is the recipe this paper follows [11].

What unites the three is a single coherence fact: the cuts of a tree are *coassociative*. Cut a tree, then cut one of the pieces; the result does not depend on the order. That fact, with the disjoint-union product and a recursively defined antipode, is what makes the trees a Hopf algebra—and it is the fact that took the bulk of the formalization.

Why this paper. Mathlib has the *target*: a `HopfAlgebra` typeclass, `MonoidAlgebra`, `TensorProduct`, and an abstract `PreLie` class that cites Chapoton–Livernet. It does not have the *object*: no rooted tree is built, no cut coproduct, no combinatorial Hopf algebra of trees. A focused search across Lean/Mathlib, the Isabelle/HOL *Archive of Formal Proofs*, and Coq/Rocq found the Connes–Kreimer / Butcher algebra absent from all of them (Section 10). This paper

`v4.30.0-rc2` and the pinned Mathlib commit `701fb6e9c3`. Reproduce with `lake exe cache get && lake env lean ConnesKreimer.lean`; the file ends with `#print axioms` for all three instances, each reporting `[propext, Classical.choice, Quot.sound]` and no `sorryAx`. The coproduct and antipode were additionally `#eval`-validated against hand-computed cuts (`ConnesKreimerEval.lean`).

supplies it—in the planar (Foissy) version, which is free of the quotient that the commutative version needs and so can be built first—and registers it as a genuine Mathlib instance.

2 The objects: trees, forests, the carrier

What, exactly, is the algebra built on?

A rooted tree, planar, is a root carrying an *ordered* list of subtrees; a forest is a list of trees. In Lean this is one nested inductive type, and the choice of `List` (rather than `Multiset`) is deliberate: a nested inductive over a quotient breaks Lean's positivity checker, whereas the `List` version compiles and supports well-founded recursion. Ordered children give the *planar* (noncommutative) algebra; the commutative one is its abelianization, deferred (Section 8).

Definition 1 (Trees, forests, carrier)**.** *`RTree` is the nested inductive `node : List RTree -> RTree`; `Forest := List RTree`; and the carrier is the free module on forests with the concatenation product,*

$$\mathcal{H} \;=\; \texttt{MonoidAlgebra}\; k\; (\texttt{FreeMonoid}\; \texttt{RTree}),$$

i.e. $\mathcal{H} = (\text{forests} \to_0 k)$ *with* $\texttt{single}(F) \cdot \texttt{single}(G) = \texttt{single}(F +\!\!+ G)$*. The unit* 1 *is the empty forest; the grafting operator* $B_+ : \text{forests} \to \text{trees}$ *is exactly the* `node` *constructor.*

The product is concatenation of forests, so $\mathcal{H}$ is noncommutative: $s \bullet t \neq t \bullet s$ as planar forests. This is the Foissy algebra. Writing $\bullet$ for the single-node tree and ℓ_2 for the two-node "cherry-stalk" $B_+(\bullet)$, the grading is by node count, with $\mathcal{H}_0 = k \cdot 1$.

3 The coproduct: cutting by recursion

How do you cut a tree—and why define it by recursion rather than by enumerating cuts?

An *admissible cut* of a tree removes a set of edges so that each path from the root to a leaf crosses at most one removed edge; it splits the tree into a pruned forest $P^c(t)$ (the parts above the cuts) and a trunk $R^c(t)$ (the part still attached to the root). The coproduct sums over all admissible cuts, including the two trivial ones:

$$\Delta(t) \;=\; t \otimes 1 + 1 \otimes t + \sum_{c \text{ proper admissible}} P^c(t) \otimes R^c(t).$$

Enumerating cuts directly is awkward to reason about. The clean definition—the one that makes coassociativity provable—is the B_+ *recursion* (the Hochschild 1-cocycle): Δ is multiplicative, $\Delta(1) = 1 \otimes 1$, and

$$\Delta(B_+ F) \;=\; B_+ F \otimes 1 + (\mathrm{id} \otimes B_+)\, \Delta F.$$

In Lean this is the mutually recursive pair `coprodTree`/`coprodForest` on the `List`-tree type, returning a list of forest pairs; forest concatenation maps to the bilinear `tmul` on tensors, and Δ is the linear extension off the forest basis (`Finsupp.linearCombination`). The recursion is computable, and was `#eval`-checked against hand-enumerated cuts.

A hand-checkable example. For the single node $\bullet$ and the two-node tree $\ell_2 = B_+(\bullet)$,

$$\Delta(\bullet) = \bullet \otimes 1 + 1 \otimes \bullet, \qquad \Delta(\ell_2) = \ell_2 \otimes 1 + 1 \otimes \ell_2 + \bullet \otimes \bullet.$$

The middle term of $\Delta(\ell_2)$ is the one nontrivial cut: snip the single edge of the cherry, the leaf $\bullet$ floats off (the pruned forest), the root $\bullet$ stays (the trunk). One can check the B_+ recursion reproduces exactly this: $\Delta(\ell_2) = \Delta(B_+\bullet) = B_+ \bullet \otimes 1 + (\mathrm{id} \otimes B_+)\Delta(\bullet) = \ell_2 \otimes 1 + (\mathrm{id} \otimes B_+)(\bullet \otimes 1 + 1 \otimes \bullet) = \ell_2 \otimes 1 + \bullet \otimes B_+(1) + 1 \otimes B_+(\bullet)$, and $B_+(1) = \bullet$, $B_+(\bullet) = \ell_2$.

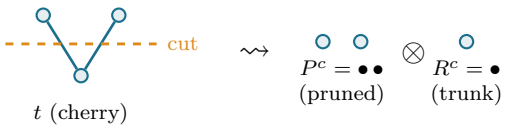


Figure 2: *An admissible cut, made visible.* The cherry tree t is a root with two leaves. Cutting *both* edges (amber) sends the two leaves into the *pruned forest* $P^c = \bullet\bullet$ and leaves the root as the *trunk* $R^c = \bullet$, contributing the term $\bullet\bullet \otimes \bullet$ to $\Delta(t)$. "Admissible" means every root-to-leaf path crosses at most one cut edge. Summing over all *five* admissible cuts of t—the empty cut $t \otimes 1$, the full cut $1 \otimes t$, the two single-edge cuts (each pruning one leaf, leaving the two-node tree ℓ_2, so $\bullet \otimes \ell_2$), and the double cut shown—gives $\Delta(t) = t \otimes 1 + 1 \otimes t + 2\,(\bullet \otimes \ell_2) + \bullet\bullet \otimes \bullet$.

The counit. ε picks out the empty-forest (degree-zero) component: $\varepsilon(1) = 1$ and $\varepsilon(F) = 0$ for any nonempty forest F. In Lean it is the algebra homomorphism $\varepsilon_{\mathrm{alg}}$ induced by the monoid hom forest $\mapsto 0$ (`FreeMonoid.lift`), i.e. the indicator of the empty forest. It is an algebra map by construction, which is what the bialgebra axioms will need.

4 Coassociativity, and the Coalgebra

Cut a tree, then cut a piece. Does the order matter?

It does not—this is the heart of the formalization, and about 80% of its effort. The subtlety the `List` encoding forces into the open is that the two iterated coproducts are not *literally* the same list of triples: they are the same *up to permutation* of the cuts. So coassociativity is first proved as a `List.Perm` statement and then lifted.

Theorem 1 (Coassociativity, in Lean 4)**.** *For every forest f, the two iterated coproducts agree as multisets of cuts,*

$$(\Delta \otimes \mathrm{id})\Delta \;=\; (\mathrm{id} \otimes \Delta)\Delta \quad \text{on } \mathcal{H},$$

and hence $(\mathcal{H}, \Delta, \varepsilon)$ is a coalgebra. The combinatorial core `coprod_coassoc` *(an equality up to* `List.Perm`*) is* `sorry`*-free with axioms* `[propext, Quot.sound]`*; lifted through the permutation-invariance of a sum of pure tensors it becomes the Mathlib linear-map equality* `coassoc_map`*, giving the instance* `instCKCoalg` *(axioms* `[propext, Classical.choice, Quot.sound]`*).*

The proof has two reusable pieces. The first is that the coproduct is *multiplicative* for the concatenation product at the combinatorial level (`coprodForest_append`: cutting a concatenation is the bilinear product `tmul` of the cuts), with `tmul` literally associative. The second, and the genuine crux, is a *Fubini* lemma: two nested `flatMap`s over cut-lists commute up to permutation

(`flatMap_comm`). With these, the tree case reduces to the B_+ cocycle: cancel the common head $[B_+F] \otimes 1 \otimes 1$, and what remains is a single application of the forest induction hypothesis after one `flatMap_comm` swap under the grafting map. The bridge to Mathlib is then short: on a basis forest both sides of the linear-map equation are sums of pure tensors over the two cut-lists, the lists are `Perm` by the combinatorial theorem, and a sum is `Perm`-invariant (`List.Perm.sum_eq`) in the commutative monoid $\mathcal{H} \otimes \mathcal{H} \otimes \mathcal{H}$.

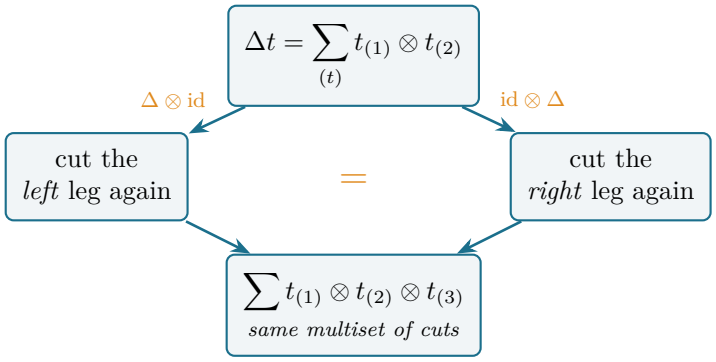


Figure 3: *The cuts cohere (coassociativity).* Cut a tree once (Δ, top), then cut one of the two resulting pieces. Cutting the left leg again ($\Delta \otimes \mathrm{id}$) and cutting the right leg again ($\mathrm{id} \otimes \Delta$) land on the *same* collection of three-way cuts—not as literally the same list (the order in which cuts are enumerated differs), but as the same *multiset*, which is exactly what the proof establishes as a `List.Perm` and then lifts to the equality of linear maps. Coassociativity is this diamond closing.

The counit laws, plainly. That ε is a left and right counit ($(\varepsilon \otimes \mathrm{id})\Delta = \mathrm{id} = (\mathrm{id} \otimes \varepsilon)\Delta$) is the statement that applying ε to one leg of every cut selects the *unique* cut with that leg empty—$(F, \varnothing)$ on the right, $(\varnothing, F)$ on the left. In Lean these are the mutual inductions `collapseR_forest`/`collapseL_forest`, and the uniqueness is the clean combinatorial identity `coprodForest_filter_emptyR` $= [(f, \varnothing)]$ (and its mirror), which also drives the antipode below.

5 The Bialgebra: product and coproduct in concert

The trees have a product (disjoint union) and a coproduct (cuts). Are they compatible?

Yes: Δ and ε are algebra homomorphisms, which is exactly the bialgebra condition.

Theorem 2 (Bialgebra, in Lean 4)**.** $(\mathcal{H}, \cdot, 1, \Delta, \varepsilon)$ *is a bialgebra:*

$$\Delta(ab) = \Delta(a)\Delta(b), \quad \Delta(1) = 1 \otimes 1, \quad \varepsilon(ab) = \varepsilon(a)\varepsilon(b), \quad \varepsilon(1) = 1.$$

sorry-free (`instCKBialg`, via `Bialgebra.mk'`), axioms `[propext, Classical.choice, Quot.sound]`.

The counit half is free: ε was defined as an algebra hom, so `map_one` and `map_mul` discharge it. The coproduct half rests on the combinatorial multiplicativity `coprodForest_append` already proved for coassociativity, repackaged into the tensor algebra (`e2_mul`, via `tmul_mul_tmul` and `single_append`) and extended from the forest basis to all of $\mathcal{H}$ by `MonoidAlgebra.induction_on`. (A Lean-specific note worth recording: the extension must use `MonoidAlgebra.induction_on`, not `Finsupp.induction_linear`; the two `add` instances on `Finsupp` versus `MonoidAlgebra` form a diamond that breaks `add_mul` under the naive induction.)

6 The antipode, and the Hopf algebra

A Hopf algebra is a bialgebra with an antipode—a convolution inverse of the identity. Where does it come from, and why is one half of its defining law so much harder than the other?

The antipode exists because the algebra is *connected graded* ($\mathcal{H}_0 = k \cdot 1$): there is a recursion that produces it. On a nonempty forest,

$$S(1) = 1, \qquad S(f) = -\sum_{(p,r)\ \text{cut},\ r\neq\varnothing} S(p)\cdot r.$$

The recursion terminates because the node count is conserved by every cut ($|p| + |r| = |f|$, `size_consForest`) and the right leg r is nonempty, so $|p| < |f|$ strictly. In Lean this is well-founded recursion on `sizeF` (`antipodeF`), extended linearly to `antipode`.

A hand-checkable example. $S(\bullet) = -\bullet$, and $S(\ell_2) = \bullet\ \bullet - \ell_2$ (the textbook values). The second: the cuts of ℓ_2 with nonempty right leg are $(\bullet, \bullet)$ and the full $(\varnothing, \ell_2)$, giving $S(\ell_2) = -(S(\bullet)\cdot\bullet + S(\varnothing)\cdot\ell_2) = -(-\bullet\cdot\bullet + \ell_2) = \bullet\,\bullet - \ell_2$. Both were `#eval`-validated, and—independently—the *other* convolution $\mathrm{id} \star S$ was checked to vanish on degree ≥ 1 and equal 1 on the unit, confirming a genuine two-sided antipode before any formal proof.

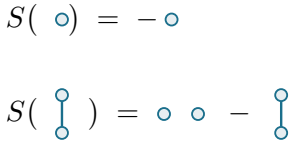


Figure 4: The antipode on small trees, drawn. *Top:* $S(\bullet) = -\bullet$. *Bottom:* $S(\ell_2) = \bullet\,\bullet - \ell_2$, where ℓ_2 is the two-node tree (a single edge) and $\bullet\bullet$ the forest of two separate nodes (the disjoint-union product $\bullet \cdot \bullet$). These are the textbook Connes–Kreimer values, reproduced here by the formalized recursion and cross-checked in Table 3.

Theorem 3 (Hopf algebra, in Lean 4)**.** *$\mathcal{H}$ is a Hopf algebra: the antipode S satisfies both*

$$\mu\,(S \otimes \mathrm{id})\,\Delta = \eta\varepsilon = \mu\,(\mathrm{id} \otimes S)\,\Delta.$$

`sorry`-free (`instCKHopf`), axioms `[propext, Classical.choice, Quot.sound]`.

The two axioms are strikingly asymmetric in cost. The *left* axiom $\mu(S \otimes \mathrm{id})\Delta = \eta\varepsilon$ is near-definitional: applying $\mu(S \otimes \mathrm{id})$ to the cut-list reassembles exactly the defining sum of S, and splitting the cuts by "empty right leg" (`coprodForest_filter_emptyR`) isolates the recursion's own equation (`leftConvF`), so the identity falls out (`leftAntipode_eq`).

The *right* axiom is the connected-graded heart, and here the formalization takes a route worth naming. Rather than re-run a (harder) right-handed recursion proof, I define a *mirror antipode* S' by the symmetric recursion $S'(f) = -\sum_{(p,r),\,p\neq\varnothing} p \cdot S'(r)$, prove its *right* convolution law by the same near-definitional argument with legs swapped (`rightAntipode'_eq`), and then prove $S = S'$ by pure algebra. The tool is Mathlib's convolution monoid `WithConv` on $\mathcal{H} \to \mathcal{H}$: the convolution product is a monoid with unit $\eta\varepsilon$, and a left inverse of id that admits a right inverse must equal it,

$$S = S \star (\mathrm{id} \star S') = (S \star \mathrm{id}) \star S' = S',$$

where the middle step is the associativity that `WithConv` supplies *for free* (`antipode_eq_antipode'`). With $S = S'$, the right axiom for S is the right axiom for S'. This sidesteps the usual induction-heavy proof of the second antipode identity entirely.

7 The certified core, and the evidence

What, concretely, has the machine checked—and how do I know the antipode is the right map and not merely a terminating recursion?

Two kinds of evidence stand behind this paper, and I give both as data. The first is the *axiom footprint*: every instance below is `sorry`-free, and `#print axioms` on each of the three (`instCKCoalg`, `instCKBialg`, `instCKHopf`) reports the identical line `[propext, Classical.choice, Quot.sound]`—the three standard axioms of classical Mathlib, no `sorryAx`, no extra postulate. The combinatorial coassociativity `coprod_coassoc` is leaner still, needing only `[propext, Quot.sound]`. The single self-contained source file builds under `lake env lean` with exit code 0 against Lean `v4.30.0-rc2` and the pinned Mathlib. Table 1 gives the development in numbers; Table 2 lists each certified ingredient.

Table 1: The development in numbers. Counts are over the three source files; "theorems/lemmas" counts all `theorem` declarations (Lean uses one keyword for both). The three headline instances are `Coalgebra`, `Bialgebra`, `HopfAlgebra`; the other two are the `CoalgebraStruct` and `HopfAlgebraStruct` carriers.

Quantity	Value
Main source `ConnesKreimer.lean` (lines)	955
Standalone coassociativity `Coassoc.lean` (lines)	242
`#eval` validation `ConnesKreimerEval.lean` (lines)	75
Theorems / lemmas	73
Definitions	21
Mathlib instances registered	5 (3 headline)
Axiom footprint	`propext, Classical.choice, Quot.sound`
Compile time (warm Mathlib cache)	≈ 20 s
Toolchain	Lean `v4.30.0-rc2`, Mathlib `701fb6e9`

The second is an *independent computational cross-check* of the antipode. The antipode is *extracted* from three data—the coproduct, the product, and the counit—by the recursion $S \star \mathrm{id} = \eta\varepsilon$; a recursion can terminate and still encode the wrong map. So the antipode was evaluated on small forests by a separate core-Lean program (no Mathlib, no shared code with the proof), and then validated by computing the *other* convolution $\mathrm{id} \star S = \sum_{(x)} x_{(1)}\, S(x_{(2)})$, which a genuine two-sided antipode must send to $\eta\varepsilon(x)$—namely 0 on every nonempty forest and 1 on the empty one. Table 3 reports the actual `#eval` output. The computed values $S(\bullet) = -\bullet$ and $S(\ell_2) = \bullet\bullet - \ell_2$ match the textbook Connes–Kreimer antipode; $S(\bullet\bullet) = \bullet\bullet$ is the consistency $S(\bullet\bullet) = S(\bullet)^2$; and the right-hand column is uniformly $\eta\varepsilon$, confirming the two-sided antipode numerically before a line of it was formally proved.

Table 2: The certified core. All are `sorry`-free; `#print axioms` on the three instances reports only `[propext, Classical.choice, Quot.sound]` (the combinatorial `coprod_coassoc` needs only `[propext, Quot.sound]`). Source: `ConnesKreimer.lean`.

Lean name	Statement	§
`coprodForest`	cut coproduct via the B_+ cocycle	3
`coprodForest_append`	Δ multiplicative (combinatorial)	4
`coprod_coassoc`	cuts cohere up to `List.Perm`	4
`coassoc_map`	$(\Delta \otimes \mathrm{id})\Delta = (\mathrm{id} \otimes \Delta)\Delta$	4
`instCKCoalg`	`Coalgebra` $k\ \mathcal{H}$	4
`Δ_mul, Δ_one`	$\Delta(ab) = \Delta a\,\Delta b$, $\Delta 1 = 1 \otimes 1$	5
`instCKBialg`	`Bialgebra` $k\ \mathcal{H}$	5
`antipodeF`	connected-graded antipode recursion	6
`leftAntipode_eq`	$\mu(S \otimes \mathrm{id})\Delta = \eta\varepsilon$	6
`rightAntipode'_eq`	$\mu(\mathrm{id} \otimes S')\Delta = \eta\varepsilon$	6
`antipode_eq_antipode'`	$S = S'$ via `WithConv`	6
`instCKHopf`	`HopfAlgebra` $k\ \mathcal{H}$	6

Table 3: Independent computational validation of the antipode (verbatim `#eval` output of `ConnesKreimerEval.lean`, core Lean, no Mathlib). Notation: $\bullet$ the single node, ℓ_2 the two-node ladder, ℓ_3 the three-node ladder, *cherry* a root with two leaves; juxtaposition is the (noncommutative) forest product, so $\ell_2\bullet \neq \bullet\ell_2$. The middle column is the extracted antipode S; the right column is the *independent* two-sided check $\mathrm{id} \star S$, which equals $\eta\varepsilon$ (here 0 for nonempty forests, 1 for $\varnothing$) exactly when S is a genuine antipode.

forest x	antipode $S(x)$	$(\mathrm{id} \star S)(x)$
$\bullet$	$-\bullet$	0
ℓ_2	$\bullet\bullet - \ell_2$	0
$\bullet\bullet$	$\bullet\bullet \quad (= S(\bullet)^2)$	0
cherry	$-\bullet\bullet\bullet + 2\,\bullet\ell_2 -$ cherry	0
ℓ_3	$-\ell_3 + \bullet\ell_2 + \ell_2\bullet - \bullet\bullet\bullet$	0
$\varnothing$	1	1

8 What I claim, and what I do not

The mathematics is classical: Connes–Kreimer, Butcher, Foissy, Grossman–Larson, and Manchon are all prior. The contribution is the formalization, and I separate the two on purpose.

What I claim as new (as formalization). We are unaware of any previous machine-checked construction of the Connes–Kreimer / Butcher / Foissy Hopf algebra of rooted trees—the cut coproduct, its coassociativity, the bialgebra compatibility, and a connected-graded antipode, assembled into a genuine Mathlib `HopfAlgebra` instance. So that the claim is reproducible rather than rhetorical, the search (run 2026-06-19) is stated explicitly. It covered: *(a)* Mathlib, full source, for the identifiers and strings `RootedTree`, `ConnesKreimer`, `Butcher`, `Foissy`, `PreLie`, and `HopfAlgebra`; *(b)* the Isabelle/HOL *Archive of Formal Proofs*, entry index and full-text, for *rooted tree*, *Connes–Kreimer*, *Butcher group*, and *Hopf algebra of trees*; and *(c)* Coq/Rocq—the opam package index, MathComp, and the Coq Package Index—for the same terms. Mathlib carries the abstract `HopfAlgebra` and `PreLie` classes but builds no rooted-tree object; the AFP and Coq

searches surfaced no combinatorial renormalization Hopf algebra of trees. This is a search result, not a theorem; a pointer to anything missed would be welcome and gladly cited.

And, plainly, what this is not. *(i)* This is the *planar* (Foissy, noncommutative) algebra, built on ordered children (`List`). The classical commutative Connes–Kreimer / Butcher algebra is its abelianization—the quotient by reordering children—which I have *not* built; the hard proof (coassociativity) is done once here on the quotient-free type, and descending to the commutative quotient is deferred follow-on work. *(ii)* I formalize the algebraic Hopf structure, not its uses: no B-series order condition, no Birkhoff/BPHZ decomposition, no pre-Lie bracket is proved here—those are the agenda (Section 9), not the contribution. *(iii)* The grading is established (node count, conserved by cuts) and used for the antipode's termination, but the full graded-connected machinery of Mathlib is not invoked; the antipode is built by explicit well-founded recursion.

9 Implications

Three kinds, in decreasing order of how firmly I can claim them.

A reusable certified core. The tree type, the cut coproduct, the Fubini/permutation machinery for coassociativity, and the connected-graded antipode (with the `WithConv` trick for the second axiom) are now machine-checked and available. This is the implication I can state most firmly: anyone formalizing tree-indexed series, combinatorial Hopf algebras, or the structures below now has a worked foundation rather than a blank page.

A foothold for what these unlock. The agenda this opens, roughly in order of effort: *B-series and the Butcher group*, hence *certified Runge–Kutta order conditions*—an applied payoff, since Mathlib has ODE existence (Picard–Lindelöf) but no order theory; the *free pre-Lie algebra* and Grossman–Larson duality, a natural Mathlib-PR target that plugs into the existing abstract `PreLie`; and, the jewel, the *Birkhoff decomposition* that realizes BPHZ renormalization as a factorization in the character group. Each is a landmark these stones could support; none is claimed here.

The mathematics, with a note of caution. This structure matters because what sits downstream of it—renormalization, numerical integration, pre-Lie geometry—matters. I am wary of borrowing that importance: certifying a foundational algebra ships no renormalized amplitude and no integrator. Its value is in certified foundations and in the growing corpus of formalized mathematics, not in immediate application; I claim the former, not the latter.

Tools. A large language model was used as a coding assistant in writing the Lean development; every statement was independently checked by the Lean kernel, and the mathematical decisions and claims are the author's.

10 Related work

The mathematics is classical: Cayley's enumeration of rooted trees [3]; Butcher's algebra of Runge–Kutta methods [2, 8]; the Connes–Kreimer Hopf algebra of renormalization [5, 9] and Brouder's identification of it with the Butcher algebra [1]; Foissy's noncommutative (planar) version [6]; Grossman–Larson's dual [7]; Chapoton–Livernet's free pre-Lie description [4]; and Manchon's

survey, whose recipe this paper follows [11]. On the formalization side, Mathlib [12] provides the `HopfAlgebra` typeclass, `MonoidAlgebra`, `TensorProduct`, and an abstract `PreLie` class citing Chapoton–Livernet—but builds no rooted tree, no cut coproduct, and no combinatorial Hopf algebra (the search supporting this is stated, with its scope, in Section 8). This development is a companion in method to the author's earlier Lean formalizations in the Godsil–Gutman / Ihara series [13].

Table 4 makes the relationship to Mathlib precise: what the library already had, what this work adds, and what remains open. Stating the last column—the structures *not* built—is as much the point as the first: it draws the boundary of the contribution honestly and names the agenda of Section 9.

Table 4: This work against Mathlib. "Abstract" classes are the general algebraic typeclasses Mathlib ships; "object" rows are the concrete rooted-tree structures. The contribution is exactly the *No* → *Yes* rows; the *No* → *No* rows are the deferred agenda (Section 9), not omissions hidden by the headline.

Structure	In Mathlib	This work
Abstract `HopfAlgebra` / `Bialgebra` / `Coalgebra` class	yes	used
Abstract `PreLie` class (cites Chapoton–Livernet)	yes	not instantiated
`MonoidAlgebra`, `TensorProduct`, `WithConv`	yes	used
Rooted trees / forests (the object)	no	**yes**
Cut coproduct Δ (B_+ cocycle)	no	**yes**
Coassociativity of the cuts	no	**yes**
`Bialgebra` instance on trees	no	**yes**
Connected-graded antipode	no	**yes**
`HopfAlgebra` instance on trees	no	**yes**
Commutative (Butcher/CK) quotient	no	no (agenda)
B-series / Runge–Kutta order conditions	no	no (agenda)
Free pre-Lie algebra (the object)	no	no (agenda)
Birkhoff / BPHZ decomposition	no	no (agenda)

11 Conclusion

Starting from the cut coproduct on rooted trees, this paper formalized, in Lean 4 / Mathlib, its coassociativity, the compatibility of the disjoint-union product, and a connected-graded antipode, assembling them into a genuine Mathlib `HopfAlgebra` instance. None of this is new mathematics; the contribution is that the Connes–Kreimer / Butcher / Foissy algebra of rooted trees is now machine-verified, in its planar form, with a reusable certified core, where a focused search found no prior formalization. The commutative quotient, B-series order conditions, the free pre-Lie algebra, and Birkhoff renormalization remain as follow-on work.

Artifacts. The Lean source (`sorry`-free, depending only on the three standard axioms) is the self-contained file `ConnesKreimer.lean`, with `Coassoc.lean` (the standalone combinatorial coassociativity) and `ConnesKreimerEval.lean` (the `#eval` validation). Verify the axiom footprint with, e.g., `#print axioms CK.instCKHopf` ⇝ `[propext, Classical.choice, Quot.sound]`. The artifact is archived at Zenodo, DOI `10.5281/zenodo.20762280`, and mirrored at https://github.com/karlesmarin/connes-kreimer-lean.

References

[1] C. Brouder, *Runge–Kutta methods and renormalization*, Eur. Phys. J. C **12** (2000), 521–534.

[2] J. C. Butcher, *An algebraic theory of integration methods*, Math. Comp. **26** (1972), 79–106.

[3] A. Cayley, *On the theory of the analytical forms called trees*, Philos. Mag. **13** (1857), 172–176.

[4] F. Chapoton and M. Livernet, *Pre-Lie algebras and the rooted trees operad*, Internat. Math. Res. Notices **2001**, no. 8, 395–408.

[5] A. Connes and D. Kreimer, *Hopf algebras, renormalization and noncommutative geometry*, Comm. Math. Phys. **199** (1998), 203–242.

[6] L. Foissy, *Les algèbres de Hopf des arbres enracinés décorés, I & II*, Bull. Sci. Math. **126** (2002), 193–239 and 249–288.

[7] R. Grossman and R. G. Larson, *Hopf-algebraic structure of families of trees*, J. Algebra **126** (1989), 184–210.

[8] E. Hairer, G. Wanner, and S. P. Nørsett, *Solving Ordinary Differential Equations I: Nonstiff Problems*, 2nd ed., Springer, 1993.

[9] D. Kreimer, *On the Hopf algebra structure of perturbative quantum field theories*, Adv. Theor. Math. Phys. **2** (1998), 303–334.

[10] L. de Moura and S. Ullrich, *The Lean 4 theorem prover and programming language*, in *Automated Deduction—CADE 28*, LNCS **12699**, Springer, 2021, 625–635.

[11] D. Manchon, *Hopf algebras, from basics to applications to renormalization*, arXiv:math/0408405 (2008).

[12] The mathlib Community, *The Lean mathematical library*, in *Proc. 9th ACM SIGPLAN Int. Conf. on Certified Programs and Proofs (CPP 2020)*, 367–381.

[13] C. Marín, *Folding Edges into Vertices: a machine-checked proof of Bass's determinant formula for the Ihara zeta in Lean 4* (2026), Zenodo DOI `10.5281/zenodo.20573120`.