`catLC`: multiscale categorical field theory for liquid crystals

[*Midterm progress report for Daniel A. Beller, last updated:* April 2, 2025]

by Alejandro Soto Franco[1]

Contents

[1]`asoto12@jhu.edu, sotofranco.math@gmail.com`

1 Introduction

Information underlies every physical model. Whether encoded in molecular configurations, order parameter fields, or large-scale defect networks, nature computes its behavior through structured data. David Deutsch argued in *The Fabric of Reality* that "every physical system has a capacity to store and process information" Deutsch (1997). In liquid crystal theory, this informational viewpoint is crucial as it connects phenomena across scales.

A fundamental tool for understanding multiscale phenomena is the **renormalization group (RG)**. RG flow describes how the parameters of a system evolve when one "zooms out" or coarse-grains the microscopic details. In liquid crystals, the RG transformation aggregates local molecular interactions into effective mesoscopic order parameters, and further into macroscopic defect configurations. This process not only reveals fixed points corresponding to phase transitions but also elucidates how local fluctuations are propagated and transformed across scales.

In my work, I propose a *multiscale categorical field theory for liquid crystals* where:

- **Objects** represent complete informational states at various scales—ranging from microscopic molecular configurations, through mesoscopic Q-tensor fields, to macroscopic defect networks.
- **Morphisms** encode the dynamical or topological transitions between these states, including the RG transformations that systematically "integrate out" degrees of freedom.
- **Functors** serve to relate and translate information between different levels of description, much like RG maps relate theories at distinct scales.

By explicitly formulating these multiscale relationships, the framework not only captures the coarse-graining procedures inherent in RG analysis but also leverages the compositional power of category theory. In this light, the RG transformation is interpreted as a functor between categories of physical theories, ensuring that the underlying informational structure is preserved as one moves from one scale to the next.

2 Renormalization Group Flow in a Categorical Framework

Renormalization group (RG) flow is central to understanding how physical systems evolve as one passes from one scale to another. In our categorical framework, the RG transformation is not merely an analytic tool but is elevated to a functorial mapping between objects representing informational states at different scales. This section elaborates on the computational aspects of RG flow and its categorification.

2.1 Conceptual Overview

Consider a microscopic state A of a liquid crystal system characterized by detailed molecular configurations. The RG flow defines a map

$$R : A \to B,$$

where B is a mesoscopic state described by an effective Q-tensor field. Repeated application of the RG transformation leads to a sequence of states

$$A \xrightarrow{R_1} B \xrightarrow{R_2} C \xrightarrow{R_3} \cdots,$$

where each R_i is a morphism in the category $\mathcal{C}$ of informational states. The composition of these morphisms,

$$R_n \circ \cdots \circ R_2 \circ R_1,$$

represents the full RG flow from the microscopic to the macroscopic level. In categorical terms, this composition is associative, ensuring that the overall transformation is well-defined independent of the grouping of intermediate steps.

2.2 Categorical Interpretation of RG Transformations

Within our category $\mathcal{C}$:

- **Objects** $A, B, C, \ldots$ denote states of the liquid crystal system at different scales. Each object is a repository of structured information.
- **Morphisms** $R_i : X \to Y$ denote RG transformations that map one scale to another. These are structure-preserving maps that integrate out microscopic fluctuations and yield effective descriptions.
- **Functoriality** ensures that the mapping from the category of microscopic theories to mesoscopic theories respects the composition of transformations. That is, if $F : \mathcal{C} \to \mathcal{D}$ is a functor representing a change of description, then

$$F(R_n \circ \cdots \circ R_1) = F(R_n) \circ \cdots \circ F(R_1).$$

2.3 Computational Framework for RG Flow

The RG transformation can be decomposed into two primary operations:

(1) **Local Averaging (Map Operation):** For each local region R in the microscopic state, compute an order parameter q_R that summarizes the local configuration. This step corresponds to integrating out high-frequency modes.

(2) **Aggregation (Reduce Operation):** Combine the locally computed order parameters q_R to form the effective mesoscopic state B. This step aggregates the local contributions into a global structure.

Mathematically, if the microscopic state is decomposed into regions $\{R_1, R_2, \ldots, R_N\}$, then the map operation yields

$$q_{R_i} = \texttt{compute\_order}(R_i) \quad \text{for } i = 1, \ldots, N,$$

and the reduce operation constructs the mesoscopic state as

$$B = \texttt{aggregate}(\{q_{R_1}, q_{R_2}, \ldots, q_{R_N}\}).$$

2.3.1 Iterative RG Transformation

An RG flow can be represented as an iterative process:

$$A^{(0)} \xrightarrow{R^{(1)}} A^{(1)} \xrightarrow{R^{(2)}} A^{(2)} \xrightarrow{R^{(3)}} \cdots,$$

where $A^{(0)}$ is the initial microscopic state, and each $A^{(k)}$ is the effective state at the kth iteration. In a computational implementation, the algorithm for a single RG step is as follows:

RG Transformation Algorithm:

(1) **Input:** State $A^{(k)}$ partitioned into local regions $\{R_1, R_2, \ldots, R_{N_k}\}$.

(2) **For each** region R_i:

   (a) Compute the local order parameter:

$$q_{R_i}^{(k)} = \texttt{compute\_order}(R_i).$$

(3) **Aggregate** the $q_{R_i}^{(k)}$ values:

$$A^{(k+1)} = \texttt{aggregate}(\{q_{R_1}^{(k)}, q_{R_2}^{(k)}, \ldots, q_{R_{N_k}}^{(k)}\}).$$

(4) **Return** the state $A^{(k+1)}$.

This algorithm, by construction, is a morphism in the category $\mathcal{C}$. Composing several such algorithms corresponds to iterating the RG transformation, which converges towards fixed points corresponding to universal behavior.

2.4 Computational Example in Rust

The following Rust code demonstrates a preliminary implementation of the RG transformation. The code captures the essential map and reduce steps, representing them as pure functions and showcasing the compositionality of the process.

```rust
#[derive(Debug)]
struct Region {
    // Represents the microscopic configuration in a local region.
    data: Vec<f64>,
}
```

```rust

#[derive(Debug)]
struct MicroscopicState {
    // The microscopic state as a collection of local regions.
    regions: Vec<Region>,
}

#[derive(Debug)]
struct OrderParameter {
    // An effective order parameter computed for a region.
    value: f64,
}

#[derive(Debug)]
struct MesoscopicState {
    // The aggregated state obtained from local order parameters.
    order_parameters: Vec<OrderParameter>,
}

/// Computes the local order parameter for a given region.
/// This function represents the "map" in the RG transformation.
fn compute_order(region: &Region) -> OrderParameter {
    let sum: f64 = region.data.iter().sum();
    let avg = sum / (region.data.len() as f64);
    OrderParameter { value: avg }
}

/// Aggregates local order parameters into a global state.
/// This function represents the "reduce" operation.
fn aggregate(order_parameters: Vec<OrderParameter>) -> MesoscopicState {
    MesoscopicState { order_parameters }
}

/// Performs a single RG transformation step,
/// mapping a microscopic state to a mesoscopic state.
fn rg_step(microscopic: &MicroscopicState) -> MesoscopicState {
    let order_parameters: Vec<OrderParameter> = microscopic.regions
    .iter()
    .map(|region| compute_order(region))
    .collect();
    aggregate(order_parameters)
}

fn main() {
    // Example: Define an initial microscopic state with several regions.
    let region1 = Region { data: vec![1.0, 2.0, 3.0] };
    let region2 = Region { data: vec![4.0, 5.0, 6.0] };
    let region3 = Region { data: vec![7.0, 8.0, 9.0] };
    let microscopic = MicroscopicState {
        regions: vec![region1, region2, region3],
    };

    // Perform a single RG transformation step.
    let mesoscopic = rg_step(&microscopic);
    println!("Mesoscopic state: {:?}", mesoscopic);
}
```

Listing 1: Rust Implementation of a Single RG Transformation Step

2.5 Iterative Flow and Fixed Points

In a fully developed RG flow, the above step is iterated:

$$A^{(0)} \xrightarrow{R} A^{(1)} \xrightarrow{R} A^{(2)} \to \cdots \to A^{(\infty)},$$

where $A^{(\infty)}$ is a fixed point of the RG transformation. In categorical terms, the fixed point is an object $F \in \mathrm{Ob}(\mathcal{C})$ such that the morphism R satisfies

$$R(F) \cong F,$$

indicating invariance under the RG flow. This invariance is crucial for understanding universality classes and phase transitions.

2.6 Functorial RG Transformations

A significant strength of the categorical perspective is the ability to interpret the RG flow as a functor:

$$\mathcal{R} : \mathcal{C}_{\text{micro}} \to \mathcal{C}_{\text{meso}},$$

which maps the category of microscopic theories to the category of effective mesoscopic theories. The functor $\mathcal{R}$ satisfies:

(a) $\mathcal{R}$ assigns to every object $A \in \mathcal{C}_{\text{micro}}$ an object $\mathcal{R}(A) \in \mathcal{C}_{\text{meso}}$.

(b) For every morphism $f : A \to B$ in $\mathcal{C}_{\text{micro}}$, $\mathcal{R}(f) : \mathcal{R}(A) \to \mathcal{R}(B)$ in $\mathcal{C}_{\text{meso}}$ such that functoriality is preserved:

$$\mathcal{R}(g \circ f) = \mathcal{R}(g) \circ \mathcal{R}(f),$$

and $\mathcal{R}(\mathrm{id}_A) = \mathrm{id}_{\mathcal{R}(A)}$.

This functorial description encapsulates the RG transformation's role as a structure-preserving map between different levels of physical description.

3 Methods

3.1 Construction

Definition 3.1 (Category)**.** A *category* $\mathcal{C}$ consists of:

(i) A collection $\mathrm{Ob}(\mathcal{C})$ of *objects*. In our application, each object represents a complete informational state of a liquid crystal system; for example, a detailed Q-tensor field, a director configuration, or an entire defect network.

(ii) For any two objects $A, B \in \mathrm{Ob}(\mathcal{C})$, a set $\mathrm{Hom}_{\mathcal{C}}(A, B)$ of *morphisms* (or arrows) that represent the transitions or processes from A to B.

(iii) A composition law: for any $f \in \mathrm{Hom}_{\mathcal{C}}(A, B)$ and $g \in \mathrm{Hom}_{\mathcal{C}}(B, C)$, there exists a composite morphism $g \circ f \in \mathrm{Hom}_{\mathcal{C}}(A, C)$. This composition is *associative*; that is, for all $f \in \mathrm{Hom}_{\mathcal{C}}(A, B)$, $g \in \mathrm{Hom}_{\mathcal{C}}(B, C)$, and $h \in \mathrm{Hom}_{\mathcal{C}}(C, D)$, one has

$$h \circ (g \circ f) = (h \circ g) \circ f.$$

(iv) For each object $A \in \mathrm{Ob}(\mathcal{C})$, there exists an *identity morphism* $\mathrm{id}_A \in \mathrm{Hom}_{\mathcal{C}}(A, A)$ satisfying

$$f \circ \mathrm{id}_A = f \quad \text{and} \quad \mathrm{id}_B \circ f = f,$$

for every $f \in \mathrm{Hom}_{\mathcal{C}}(A, B)$.

Axiom 3.1 (Informational Conjecture)**.** Every object in $\mathcal{C}$ serves as a repository of information. In the context of liquid crystal physics, an object may encode a microscopic molecular configuration, a mesoscopic Q-tensor field, or a macroscopic defect network, thereby capturing the state of the system at a given scale.

Remark 1. Morphisms in the category represent the dynamical or topological transformations between informational states. The composition of these morphisms mirrors the sequential processing of information in physical systems. This intrinsic compositionality is essential in bridging scales from microscopic details to macroscopic behavior.

Theorem 3.1 (Associativity of Compositional Transformations)**.** *Let* $f \in \mathrm{Hom}_{\mathcal{C}}(A, B)$*,* $g \in \mathrm{Hom}_{\mathcal{C}}(B, C)$*, and* $h \in \mathrm{Hom}_{\mathcal{C}}(C, D)$*. Then*

$$h \circ (g \circ f) = (h \circ g) \circ f.$$

Proof. The associativity of composition is a fundamental property in the definition of a category. Hence, by definition, the composite $h \circ (g \circ f)$ must equal $(h \circ g) \circ f$ for any choice of f, g, and h in $\mathcal{C}$. □

Example 1: Liquid Crystal States as Objects and Transitions as Morphisms

Consider a liquid crystal system with the following representations:

- Let A be a microscopic state characterized by a detailed molecular configuration.
- Let B be a mesoscopic state described by a Q-tensor field.
- Let C be a macroscopic state corresponding to a network of topological defects.

A morphism $f : A \to B$ may represent a coarse-graining process that aggregates local molecular information into a continuum order parameter. Subsequently, a morphism

> $g : B \to C$ can describe the evolution of defects arising from variations in the Q-tensor field. The composite morphism $g \circ f : A \to C$ encapsulates the transition from the microscopic to the macroscopic scale, preserving the informational content throughout.

In the categorical framework for liquid crystal modeling, the coarse-graining process can be viewed as a pure function that maps a microscopic state to a mesoscopic state. This process is conceptually analogous to a pipeline in functional programming, where each function is pure (i.e., without side effects) and the overall transformation is realized via function composition.

Generic "Upward" Scaling Algorithm:

```
(1) Input: A microscopic state $A$ consisting of local regions $R_1, R_2, \ldots, R_N$.
(2) For each local region $R \subset A$:
    (a) Compute the local order parameter via statistical averaging:
            $q_R = \texttt{compute\_local\_order\_parameter}(R)$.
(3) Collect all computed values into the set $\{q_{R_1}, q_{R_2}, \ldots, q_{R_N}\}$.
(4) Aggregate these elements to obtain the mesoscopic state $B$ (i.e., a $Q$-tensor field).
(5) Return $B$.
```

Remark 2. Viewed from a computer science perspective, this algorithm can be decomposed into two main operations:

- A `map` operation that transforms each local region R into a Q-tensor element q_R.
- A `reduce` (or aggregation) operation that combines all the q_R into the global mesoscopic state B.

This is very much in the spirit of functional programming languages such as Haskell or Rust, where the emphasis is on immutability and function composition.

The first pass at this algorithm in Rust came out as:

```rust
#[derive(Debug)]
struct Region {
    // Data representing the microscopic configuration in a local region,
    // e.g., molecular orientations, positions, etc.
    data: Vec<f64>,
}

#[derive(Debug)]
struct MicroscopicState {
    // The microscopic state is a collection of regions.
    regions: Vec<Region>,
}

#[derive(Debug)]
struct QTensor {
    // A simplified representation of a Q-tensor; in practice,
    // this could be a matrix or higher-dimensional array.
    values: Vec<f64>,
}

```

```rust
#[derive(Debug)]
struct MesoscopicState {
    // The mesoscopic state aggregates the local Q-tensors.
    q_tensors: Vec<QTensor>,
}

/// Computes the local order parameter for a given region.
/// This function performs statistical averaging over the data in the region.
fn compute_local_order_parameter(region: &Region) -> QTensor {
    let sum: f64 = region.data.iter().sum();
    let avg = sum / (region.data.len() as f64);
    // For demonstration, we simply return a QTensor with a single averaged
        value.
    QTensor { values: vec![avg] }
}

/// Combines a vector of Q-tensors into a mesoscopic state.
/// This could involve additional processing in a more complex implementation.
fn combine_q_tensors(q_tensors: Vec<QTensor>) -> MesoscopicState {
    MesoscopicState { q_tensors }
}

/// Coarse-graining function: maps a microscopic state to a mesoscopic state.
fn coarse_grain(microscopic: &MicroscopicState) -> MesoscopicState {
    let q_tensors: Vec<QTensor> = microscopic.regions
    .iter()
    .map(|region| compute_local_order_parameter(region))
    .collect();
    combine_q_tensors(q_tensors)
}

fn main() {
    // Example: Create a dummy microscopic state with two regions.
    let region1 = Region { data: vec![1.0, 2.0, 3.0] };
    let region2 = Region { data: vec![4.0, 5.0, 6.0] };
    let microscopic = MicroscopicState {
        regions: vec![region1, region2],
    };

    // Perform the coarse-graining process.
    let mesoscopic = coarse_grain(&microscopic);
    println!("Mesoscopic state: {:?}", mesoscopic);
}
```

Listing 2: Preliminary Rust Code for Coarse-Graining Liquid Crystal Configurations

Remark 3. The Rust implementation demonstrates how the coarse-graining process can be expressed in a functional style:

- The use of `iter().map()` applies the `compute_local_order_parameter` function to each region in an immutable fashion.
- The aggregation of results is performed via `collect()`, reinforcing the concept of pure functional transformation.

This design not only aligns with categorical thinking—where processes (morphisms) compose to form a well-defined transformation—but also highlights the utility of modern programming paradigms in modeling complex physical systems.

References

Baez, J. C. and Stay, M. (2011). Physics, topology, logic and computation: A rosetta stone. In *New Structures for Physics*, pages 95–172. Springer.

Deutsch, D. (1997). *The Fabric of Reality.* Penguin Books, London.

Feynman, R. P. (1965). *The Character of Physical Law.* MIT Press, Cambridge, MA.