Collimator Roadmap

This document tracks potential improvements, new features, code cleanup opportunities, and technical debt for the Collimator profunctor optics library.

Feature Proposals

[Priority: High] Indexed Optics

Description: Add indexed variants of optics that carry position information through the traversal. This would enable operations like `itraversed` that provide both index and value.

Rationale: Indexed optics are a powerful pattern from Haskell's `lens` library that allows users to access element positions during traversals. The `examples/IndexedOptics.lean` file exists as a placeholder, suggesting this is a desired feature.

Affected Files: - New: `Collimator/Indexed.lean` or `Collimator/Optics/Indexed.lean` - New: `Collimator/Indexed/ILens.lean`, `ITraversal.lean`, etc. - Modify: `Collimator/Instances.lean` (add indexed versions of `traversed`) - New: `CollimatorTests/IndexedTests.lean`

Estimated Effort: Large

Dependencies: None

[Priority: High] Optic for HashMap/AssocList Access

Description: Provide `at` and `ix` optics for key-value access, similar to Haskell's `at :: Index m => Index m -> Lens' m (Maybe (IxValue m))` and `ix :: Ixed m => Index m -> Traversal' m (IxValue m)`.

Rationale: The `examples/JsonLens.lean` manually implements `field` and `index` as `AffineTraversal'`. These patterns are common enough to warrant first-class support with proper abstractions.

Affected Files: - New: `Collimator/At.lean` or extend `Collimator/Instances.lean` - Modify: `Collimator/Prelude.lean` (re-export) - New: Tests for HashMap, AssocList, Array access patterns

Estimated Effort: Medium

Dependencies: None

[Priority: Medium] Getter and Review Optic Types

Description: Add explicit `Getter` and `Review` optic types to complement the existing hierarchy. Currently, read-only access uses `Fold` and construction uses prisms, but dedicated types would improve clarity and type inference.

Rationale: The optic hierarchy is nearly complete but missing these two important types. `Getter s a` is a read-only lens, and `Review t b` is a write-only prism. Having these would complete the subtyping lattice shown in the tracing/command output.

Affected Files: - Modify: `Collimator/Optics.lean` (add `Getter`, `Review` structures) - Modify: `Collimator/Combinators.lean` (add conversion functions) - Modify: `Collimator/Exports.lean` (re-export) - New: `CollimatorTests/GetterReviewTests.lean`

Estimated Effort: Medium

Dependencies: None

[Priority: Medium] Plated and Recursive Structure Traversals

Description: Implement `Plated` typeclass for recursive data structure traversals, enabling operations like `cosmos`, `para`, and `transform` for generic recursion.

Rationale: Recursive traversals are essential for AST manipulation, tree transformations, and similar use cases. The `examples/TreeTraversal.lean` example demonstrates manual tree traversal; `Plated` would generalize this.

Affected Files: - New: `Collimator/Plated.lean` - New: Tests and examples

Estimated Effort: Large

Dependencies: None

[Priority: Medium] Bifunctor Optics (Each, Both)

Description: Add `each` and `both` combinators for traversing bifunctor structures (tuples, Either-like types).

Rationale: Common patterns like `(a, a) & both %~ f` are currently not ergonomic. The `Collimator/Instances.lean` has `Prod` lenses but not a unified `both` traversal.

Affected Files: - Modify: `Collimator/Instances.lean` (add `both` for `Prod`) - Modify: `Collimator/Combinators.lean` (add generic `each` if possible) - Add tests

Estimated Effort: Small

Dependencies: None

[Priority: Medium] Derive Prisms Macro

Description: Add a `makePrisms` command similar to `makeLenses` that automatically generates prisms for inductive type constructors.

Rationale: The `makeLenses` command in `Collimator/Derive/Lenses.lean` is well-implemented and useful. A corresponding `makePrisms` would complete the derive story for sum types.

Affected Files: - New: `Collimator/Derive/Prisms.lean` - Modify: `Collimator/Prelude.lean` (import) - New: Tests

Estimated Effort: Medium

Dependencies: None

[Priority: Low] ReaderT/StateT Optic Integration

Description: Extend `Collimator/Integration.lean` with monad transformer zoom patterns that work seamlessly with Lean 4's effect system.

Rationale: The current `Integration.lean` provides basic `StateM` and `ReaderM` utilities. More sophisticated patterns like nested zoom and optic-based `local` would improve ergonomics in monadic code.

Affected Files: - Modify: `Collimator/Integration.lean` - Add examples and tests

Estimated Effort: Medium

Dependencies: None

[Priority: Low] Property-Based Testing with Plausible

Description: Add property-based law verification using the `plausible` library, complementing the runtime law checks in `Collimator/Debug/LawCheck.lean`.

Rationale: Runtime law checking is useful but not as thorough as QuickCheck-style property testing. The workspace already has `plausible` as a dependency in other projects.

Affected Files: - Modify: `lakefile.lean` (add plausible dependency) - New: `CollimatorTests/PropertyTests.lean` - Modify: `Collimator/Testing.lean` (integrate with existing framework)

Estimated Effort: Medium

Dependencies: `plausible` library

Code Improvements

[Priority: High] Complete WIP Theorems in Equivalences.lean.wip

Current State: The file `Collimator/Theorems/Equivalences.lean.wip` contains incomplete proofs for profunctor/van Laarhoven equivalence theorems.

Proposed Change: Fix the syntax issues and complete the remaining axiomatized theorems where possible. The current `Equivalences.lean` already has 5 proven theorems but leaves 4 as axioms due to parametricity requirements.

Benefits: Stronger formal guarantees, reduced axiom count, better documentation of what's provable vs. fundamentally requires parametricity.

Affected Files: - `Collimator/Theorems/Equivalences.lean.wip` (fix and integrate) - `Collimator/Theorems/Equivalenc` (potentially merge improvements)

Estimated Effort: Large

[Priority: High] Replace Axioms with Proofs in Core.lean

Current State: The file `Collimator/Core.lean` contains two axioms at lines 277 and 331: `- axiom instLawfulStrongArrow : LawfulStrong (fun a b : Type u => a -> b) - axiom instLawfulChoiceArrow : LawfulChoice (fun a b : Type u => a -> b)`

Proposed Change: Replace these axioms with actual proofs. The laws should be provable by straightforward case analysis and function extensionality.

Benefits: Eliminates axioms from the core module, improves trustworthiness of the library.

Affected Files: - `Collimator/Core.lean`

Estimated Effort: Medium

[Priority: Medium] Improve Type Inference with Optic Subtyping

Current State: The `Collimator/Theorems/Subtyping.lean` file defines coercion functions between optic types, but type inference can still be challenging in some compositions.

Proposed Change: Add more `Coe` instances and potentially use Lean 4's instance priorities to improve automatic subtyping. Consider adding helper functions that guide type inference.

Benefits: Better ergonomics, less explicit type annotation needed.

Affected Files: - `Collimator/Theorems/Subtyping.lean` - `Collimator/Combinators.lean` (add inference helpers)

Estimated Effort: Medium

[Priority: Medium] Consolidate Redundant Traversal Definitions

Current State: Multiple places define traversal-like patterns: - `List.walkMon` in tests - `traverseList'Mon` in `Collimator/Theorems/TraversalLaws.lean` - `traversed` in `Collimator/Instances.lean`

Proposed Change: Consolidate to a single canonical implementation with the lawful instance proven once and reused.

Benefits: Reduces duplication, single source of truth for lawfulness proofs.

Affected Files: - `Collimator/Instances.lean` - `Collimator/Theorems/TraversalLaws.lean` - `CollimatorTests/TraversalTests.lean`

Estimated Effort: Small

[Priority: Medium] Add Documentation for Core Profunctor Abstractions

Current State: The `Collimator/Core.lean` file has module-level docs but individual definitions could use more detailed documentation, especially regarding when to use each profunctor class.

Proposed Change: Add comprehensive docstrings to `Profunctor`, `Strong`, `Choice`, `Wandering`, and `Closed` classes with examples and guidance.

Benefits: Better developer experience, easier onboarding.

Affected Files: - `Collimator/Core.lean`

Estimated Effort: Small

[Priority: Medium] Optimize Traversal Performance

Current State: Traversals create intermediate structures when composing multiple traversals. For example, `over' (t1 . t2) f` may not fuse efficiently.

Proposed Change: Add fusion rules or use stream fusion techniques to optimize composed traversals. Consider adding `@[inline]` and `@[specialize]` attributes strategically.

Benefits: Better runtime performance for complex optic compositions.

Affected Files: - `Collimator/Optics.lean` - `Collimator/Combinators.lean`

Estimated Effort: Medium

[Priority: Low] Unify Operators Module Organization

Current State: The `Collimator/Operators.lean` file (427 lines) contains all operators and is well-organized but quite long.

Proposed Change: Consider splitting into logical groups (viewing operators, modification operators, composition operators) or adding section markers for navigation.

Benefits: Easier maintenance and navigation.

Affected Files: - `Collimator/Operators.lean`

Estimated Effort: Small

[Priority: Low] Add Benchmarks

Current State: No performance benchmarks exist for comparing optic operations against manual alternatives.

Proposed Change: Create a benchmark suite comparing: - Optic-based access vs. direct field access - Composed traversals vs. manual recursive functions - Different traversal strategies (e.g., `each` vs. manual folds)

Benefits: Performance visibility, regression detection.

Affected Files: - New: `bench/` directory with benchmark code

Estimated Effort: Medium

Code Cleanup

[Priority: High] Complete or Remove WIP Files

Issue: Two WIP files exist that are not integrated: - `Collimator/Theorems/Equivalences.lean.wip` - `examples/ToolingDemo.lean.wip`

Location: Project root

Action Required: 1. For `Equivalences.lean.wip`: Either fix and merge into main `Equivalences.lean` or document why it's incomplete 2. For `ToolingDemo.lean.wip`: References non-existent modules (`Collimator.Commands`, `Collimator.Tracing`). Either implement these modules or remove the example

Estimated Effort: Medium (depends on whether to fix or remove)

[Priority: High] Fix Missing Module References in ToolingDemo.lean.wip

Issue: The file `examples/ToolingDemo.lean.wip` imports: - `Collimator.Commands` (does not exist) - `Collimator.Tracing` (does not exist)

Yet `Collimator/Commands.lean` (124 lines) and `Collimator/Tracing.lean` (540 lines) do exist in the source.

Location: `examples/ToolingDemo.lean.wip`

Action Required: 1. Verify the imports match actual module paths 2. Rename to `.lean` if the demo works 3. Or document what's missing

Estimated Effort: Small

[Priority: Medium] Remove Unused Concrete Profunctor: Costar

Issue: The `Collimator/Concrete/Costar.lean` file defines `Costar` profunctor but it may not be used anywhere in the library.

Location: `Collimator/Concrete/Costar.lean`

Action Required: 1. Search for usages of `Costar` in the codebase 2. If unused, either document its intended purpose or remove it 3. If used, ensure it's properly exported

Estimated Effort: Small

[Priority: Medium] Standardize Error Messages in Derive Macros

Issue: The `makeLenses` command in `Collimator/Derive/Lenses.lean` has good error messages but could use consistent formatting and potentially helpful suggestions.

Location: `Collimator/Derive/Lenses.lean` lines 196-228

Action Required: 1. Review error message format for consistency 2. Consider adding `did you mean?` suggestions 3. Ensure all error paths have helpful messages

Estimated Effort: Small

[Priority: Medium] Add Module-Level Documentation

Issue: Several modules lack module-level documentation: - `Collimator/Exports.lean` - `Collimator/Concrete/*.lean` (some have docs, some don't) - Example files

Location: Various

Action Required: Add `/-! ... -/` documentation blocks at the top of each module explaining its purpose, contents, and usage.

Estimated Effort: Medium

[Priority: Low] Clean Up Test File Organization

Issue: Tests are well-written but some test files are very long (e.g., `TraversalTests.lean` at 1183 lines).

Location: `CollimatorTests/`

Action Required: 1. Consider splitting large test files by category (e.g., `TraversalTests/Laws.lean`, `TraversalTests/Effectful.lean`) 2. Extract common test utilities to a shared module 3. Ensure consistent naming conventions

Estimated Effort: Small

[Priority: Low] Upgrade Lean Version Consistency

Issue: The project uses Lean 4.26.0 via mathlib dependency. Should evaluate upgrading to newer versions.

Location: `lakefile.lean`, `lean-toolchain`

Action Required: 1. Evaluate compatibility with newer Lean versions 2. Update mathlib dependency if stable 3. Test all modules after upgrade

Estimated Effort: Medium (depends on breaking changes)

[Priority: Low] Review and Update Comments/TODOs

Issue: Some files may contain outdated comments or implicit TODOs.

Location: Various source files

Action Required: 1. Search for `TODO`, `FIXME`, `HACK` comments 2. Either address or document as known limitations 3. Update any stale comments referencing old designs

Estimated Effort: Small

API Enhancements

[Priority: High] Add More Fold Combinators

Description: Expand `Fold` operations to include more common patterns from Haskell's lens library.

Current API: - `toList`, `toListOf`, `foldMap`, `sumOf`, `lengthOf` - `anyOf`, `allOf`, `firstOf`, `lastOf`

Proposed Additions: - `findOf` - find first element matching predicate - `elemOf` - check if element exists - `nullOf` - check if fold is empty - `minimumOf`, `maximumOf` - with `Ord` constraint - `foldl'Of`, `foldr'Of` - strict folds

Affected Files: - `Collimator/Combinators.lean` - `Collimator/Exports.lean` - Tests

Estimated Effort: Medium

[Priority: Medium] Add Setter Combinators

Description: Add more setter-specific combinators for common modification patterns.

Proposed Additions: - `mapped` - setter for functor contents - `setting` - create setter from modification function - `assign` - for use with State monad - `+=`, `-=`, `*=` operators for numeric fields

Affected Files: - `Collimator/Combinators.lean` - `Collimator/Operators.lean` - Tests

Estimated Effort: Medium

[Priority: Medium] Improve Prism Ergonomics

Description: The current prism API requires manual splitting into Sum types. Add helper constructors for common patterns.

Proposed Changes: - Add `prismFromOption : (s -> Option a) -> (a -> s) -> Prism' s a` (already exists as `prismFromPartial`) - Rename `prismFromPartial` to `prism'` for consistency with Haskell - Add `only : a -> Prism' a ()` for exact value matching - Add `nearly : a -> (a -> Bool) -> Prism' a ()` for predicate matching

Affected Files: - `Collimator/Optics.lean` - `Collimator/Helpers.lean` - `Collimator/Exports.lean`

Estimated Effort: Small

[Priority: Low] Add Optic Pretty Printing

Description: Implement `Repr` or custom pretty printing for optic types to aid debugging.

Rationale: Currently optics are functions and don't print meaningfully. Named optics with metadata would improve debugging experience.

Affected Files: - New: Potentially `Collimator/Debug/Pretty.lean` - Modify: Optic type definitions to carry optional metadata

Estimated Effort: Medium

Summary Statistics

Category	High	Medium	Low	Total
Feature Proposals	2	5	2	9
Code Improvements	2	5	2	9
Code Cleanup	2	3	3	8
API Enhancements	1	2	1	4
Total	**7**	**15**	**8**	**30**

Contributing

When working on items from this roadmap:

1. Check if an issue already exists for the work
2. Create a branch with a descriptive name (e.g., `feature/indexed-optics`)
3. Add tests for new functionality
4. Update documentation as needed
5. Run `lake build && lake test` before submitting

For questions or discussion about roadmap items, please open an issue.