

Hylograph: The Interactive Guide

Status: active **Category:** plan **Created:** 2026-01-28 **Tags:** hylograph, hats, tour, demo, visualization, ast-builder **Supersedes:** Merges concepts from two prior plans (retained for reference)

Prior Documents

This plan unifies two complementary approaches:

- 1. **AST Builder Redesign** ([ast-builder-redesign.md](#)) Focus: Internal mechanics — enumxassembly matrix, Fold node visualization, type flow
- 2. **Tour Structures Page** ([hylograph-tour-structures-page.md](#)) Focus: External inputs — Map, Parser, Free as higher-order structures to visualize

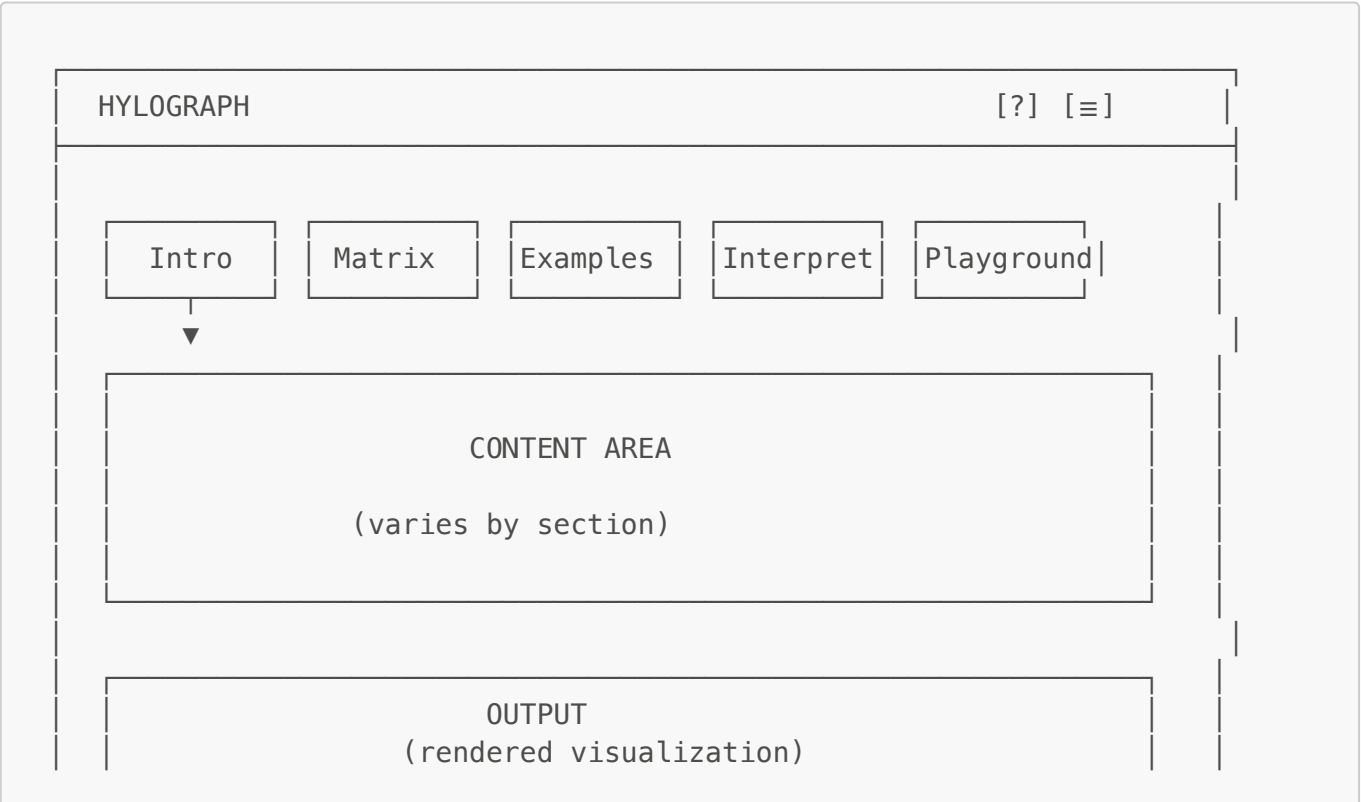
Both remain valid if we decide to split the concerns. This document explores what a unified app would look like.

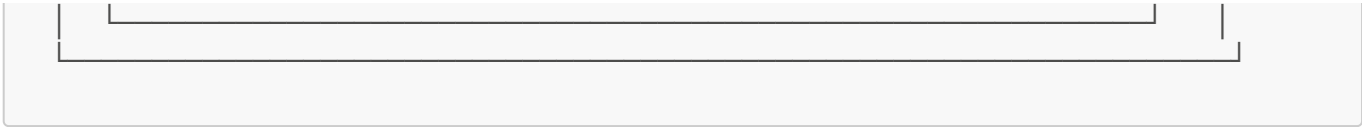
Vision

A single interactive application that serves as both **demonstration** and **documentation** of Hylograph/HATS. The demo IS the textbook.

Core message: "Pass structure in, get visualization out. The hylo doesn't care what shape — it sees trees all the way down."

Application Structure





Navigation: horizontal tabs or step-through with prev/next.

Section 1: Introduction

Purpose: Hook the reader, establish the core concept.

Content:

- Brief text: "Hylograph turns data structures into pictures through a single abstraction: the hylomorphism."
- Animated diagram: structure unfolds (ana), then folds into visualization (cata)
- The Ana/Cata dancing sisters logo (if ready)
- No jargon yet — just "unfold your data, fold into a picture"

Interactive element:

- Simple array → bar chart transformation
- User can edit the array, see the chart update
- Teaser: "What if the input was a Map? A parser? A program?"

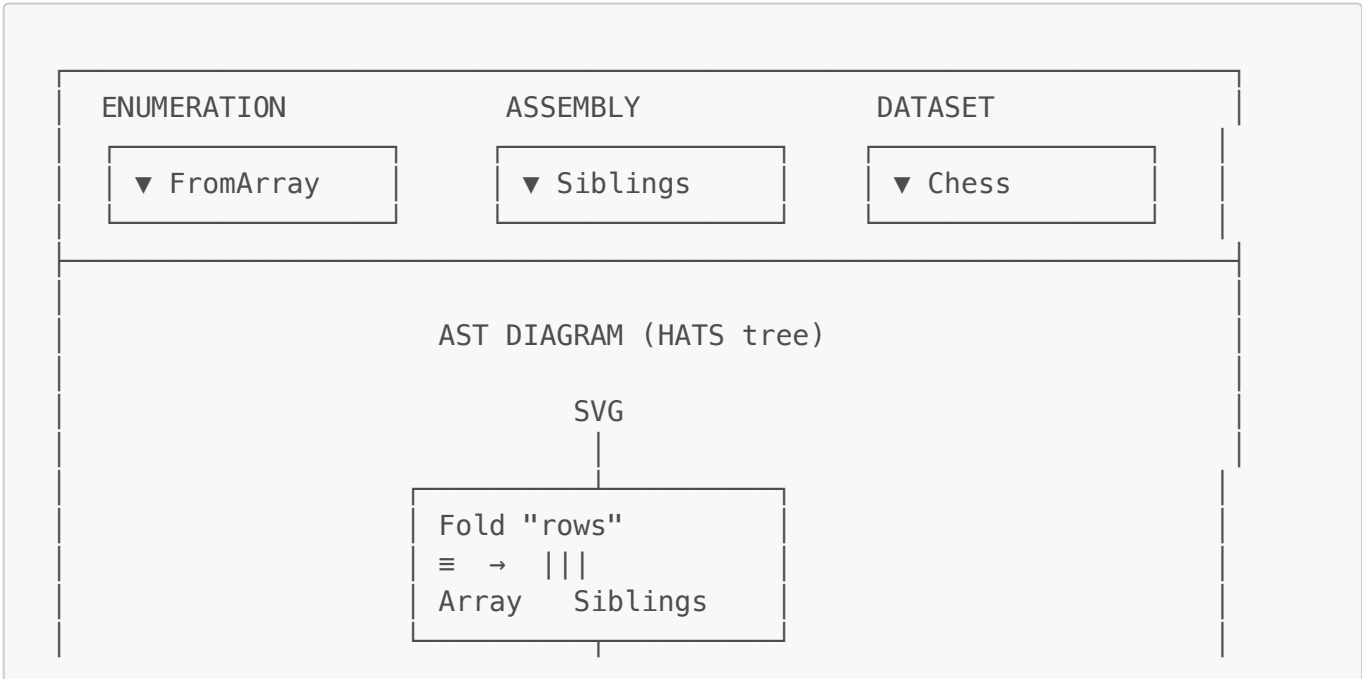
Length: One screen, minimal scrolling.

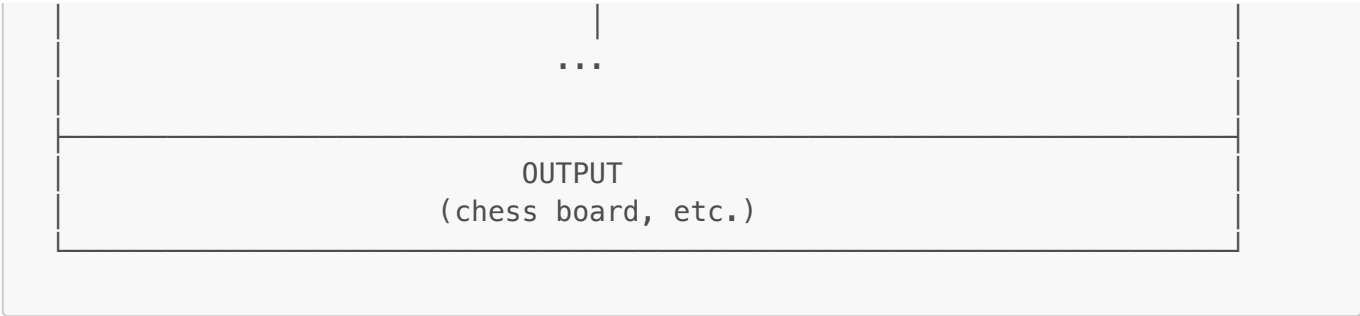
Section 2: The Matrix

Purpose: Explain the enumxassembly insight that makes HATS powerful.

Source: [ast-builder-redesign.md](#)

Layout:





Key interactions:

- 1. Change Enumeration dropdown → AST updates, incompatible datasets gray out
- 2. Change Assembly dropdown → AST updates, output structure changes
- 3. Hover on Fold node → show template, ghost repetition
- 4. Click Fold node → detail view with generated code

Teaching points:

- Enumeration and assembly are orthogonal choices
- The NxM matrix of combinations
- Type safety demonstrated by graying, not explained

Optional: Matrix thumbnail grid showing all valid combinations.

Section 3: Examples

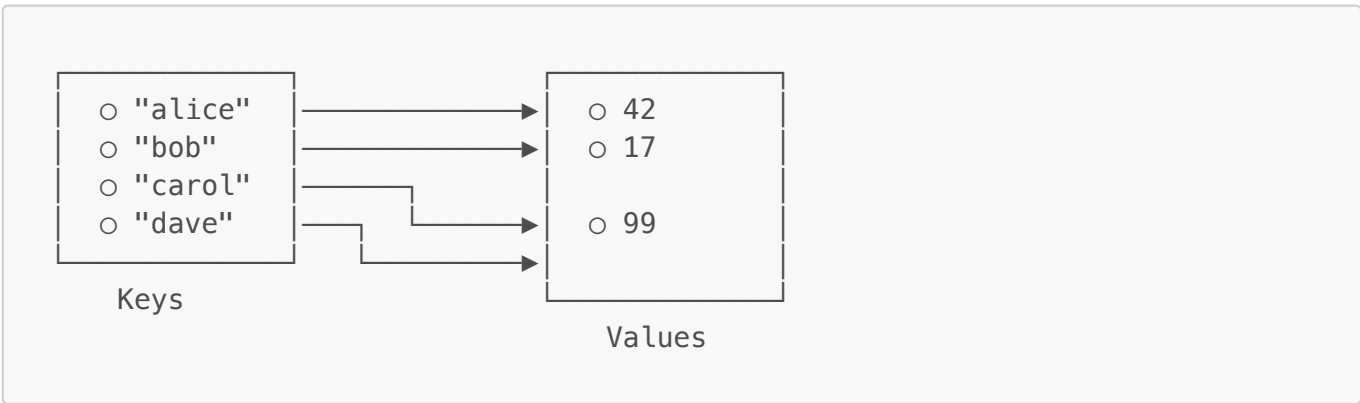
Purpose: Show the power of Hylograph on real, higher-order structures.

Source: [hylograph-tour-structures-page.md](#)

Three sub-sections, navigable as tabs or cards:

3a: Maps — The Textbook Function Diagram

What it shows: Map *k v* as domain → range with arrows.



Interactive:

- Edit the map entries
- See value sharing (carol and dave → 99)
- Toggle: show HATS tree that produces this

Code snippet:

```

userAges :: Map String Int
userAges = Map.fromFoldable [ "alice" /\ 42, "bob" /\ 17, ... ]

visualize mapDiagram userAges

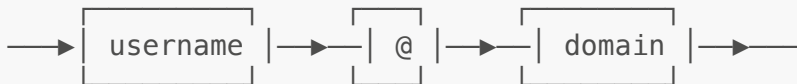
```

Teaching point: Lowers friction to visualizing Maps. Which lowers friction to using Maps.

3b: Parsers — Railroad Diagrams

What it shows: Parser combinator → railroad/syntax diagram.

emailParser:

**Interactive:**

- Select from example parsers (email, URL, arithmetic expression)
- See railroad diagram update
- Toggle: show HATS tree
- Advanced: edit parser definition, see diagram update

Code snippet:

```

emailParser :: Parser String Email
emailParser = do
  user <- username
  _ <- char '@'
  ...

visualize railroadDiagram emailParser

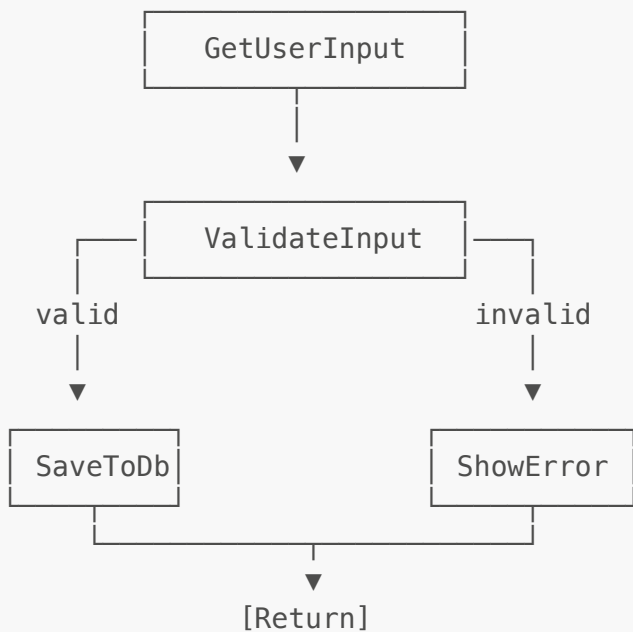
```

Teaching point: Parser structure maps directly to railroad semantics. Documentation stays in sync.

This is the "wow" demo — solves a real problem people have.

3c: Free Monads — Flowcharts

What it shows: Free f a program as flowchart.

**Interactive:**

- Select from example programs
- See flowchart update
- Toggle: show HATS tree
- Highlight: click flowchart node, see corresponding code

Code snippet:

```

myProgram :: App Unit
myProgram = do
  input <- getUserInput
  valid <- validateInput input
  if valid then saveToDb (parse input)
    else showError "Invalid"

visualize flowchart myProgram

```

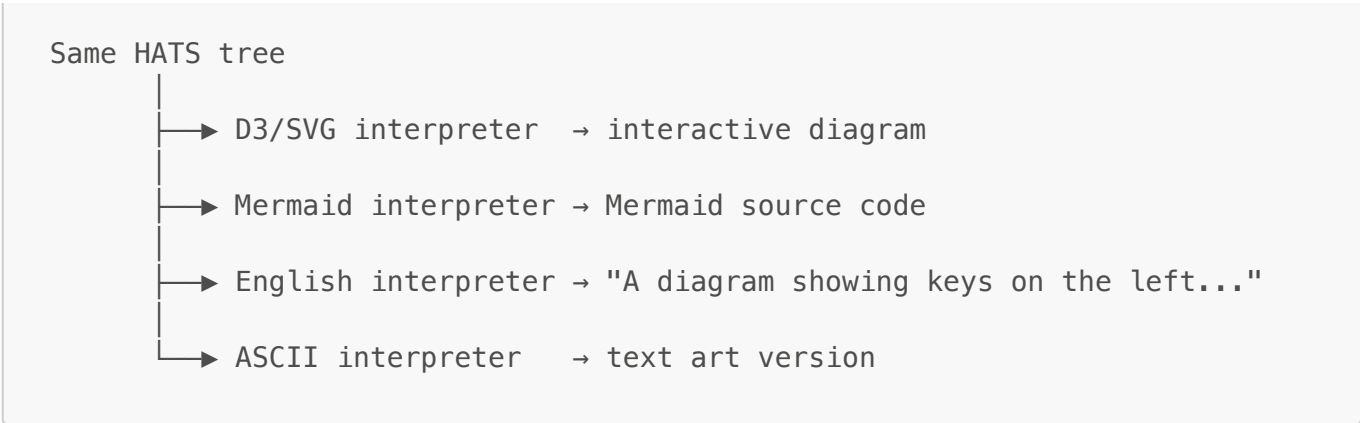
Teaching point: Free monads are "ASTs of your DSL" — this makes that literal.

Section 4: Interpreters

Purpose: Show that the same HATS tree can produce different outputs.

Content:

Take one of the examples (say, the Map diagram) and show:



Interactive:

- Dropdown to select interpreter
- Output area shows result of that interpreter
- HATS tree stays the same (highlight this)

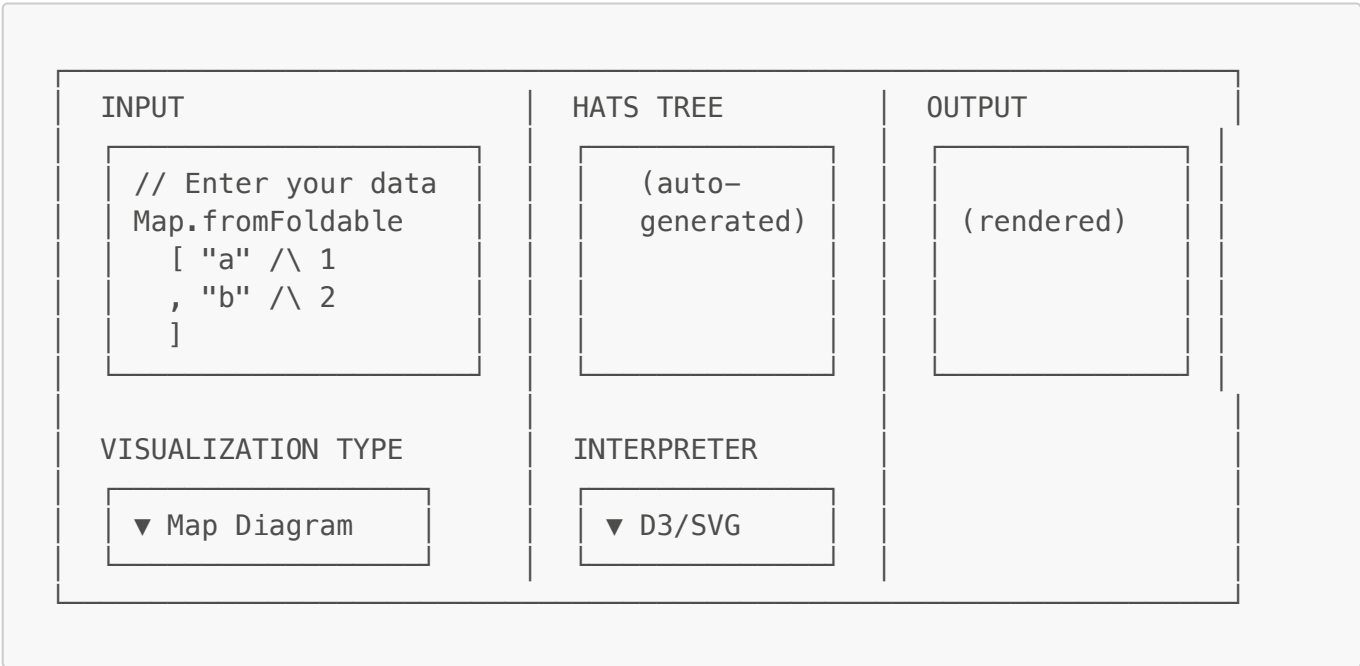
Teaching point: "The fold doesn't care what it's folding into." The algebra is swappable.

Accessibility angle: English interpreter enables screen reader descriptions generated from the same spec as the visual.

Section 5: Playground

Purpose: Let users explore freely.

Layout:



Features:

- Code editor for input data
- Dropdown for visualization type (map diagram, bar chart, tree, etc.)
- Dropdown for interpreter

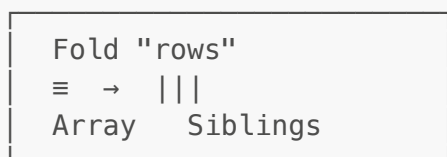
- Live HATS tree view (optional toggle)
- Shareable URLs for configurations

Stretch goal: "Bring your own structure" — define a custom pattern functor and visualize it.

Visual Design Notes

Fold Node Rendering (from ast-builder-redesign)

Compound glyph with icons:



```

Fold "rows"
≡ → |||
Array  Siblings
  
```

Icon vocabulary:

- ≡ or [...] = Array
- △ = Tree
- ○ = Graph
- ||| = Siblings
- □ = Nested
- ■ = GroupedBy

Color Palette

Consider using the Ana/Cata logo colors:

- Black background for output area
- Gold accents for interactive elements
- White/cream for content areas

Or: neutral palette that lets visualizations shine.

Typography

Monospace for code, clean sans-serif for UI, distinctive display font for "HYLOGRAPH" title.

Implementation Approach

Phase 1: Foundation

1. Set up app shell with navigation
2. Port Matrix section from existing AST Builder
3. Basic output rendering

Phase 2: Examples

4. Implement Map diagram example
5. Implement Railroad diagram example (high impact)
6. Implement Free flowchart example

Phase 3: Polish

7. Add Interpreters section
8. Build Playground
9. Animations and transitions
10. Mobile/responsive considerations

Technology

- **Framework:** Halogen (consistent with PSD3 ecosystem)
 - **Rendering:** HATS → D3 interpreter for visualizations
 - **Code editing:** CodeMirror or Monaco (for Playground)
 - **Hosting:** Static site, could be part of main PSD3 site
-

Open Questions

1. **Parser library:** Which to use for the Railroad demo? Registry parser-combinators? Custom minimal one for pedagogical clarity?
 2. **Free monad DSL:** Use a realistic DSL or a pedagogical one? Tradeoff: relatable vs. simple.
 3. **Progressive disclosure:** How much do we show upfront vs. on hover/click? The Matrix section could overwhelm.
 4. **Code visibility:** Always show code? Toggle? Only in Playground?
 5. **HATS tree visibility:** Central feature or hidden detail? The ast-builder-redesign centers it; the tour examples could hide it.
 6. **Naming:** "Hylograph Guide", "Hylograph Interactive", "The Hylograph Tour", just "Hylograph"?
 7. **Branding:** Use the Ana/Cata logo? The Afrofuturist Escher aesthetic for backgrounds/accents?
-

Success Criteria

- New user can understand enum×assembly matrix in 2 minutes
 - "Parser → Railroad" demo elicits "oh, that's cool"
 - Type safety is felt (graying) not explained (text)
 - Same HATS tree → multiple interpreters is viscerally clear
 - Playground enables genuine exploration, not just canned examples
 - Works as both a demo to show others and a reference to return to
-

Relationship to Other Docs

Document	Status	Notes
ast-builder-redesign.md	referenced	Matrix section draws heavily from this
hylograph-tour-structures-page.md	referenced	Examples section draws from this
hats-existential-design.md	background	Technical foundation for HATS
hats-halogen-integration.md	implementation	How to wire HATS into Halogen

Appendix: The Dancing Sisters

The Ana/Cata logo concept:

- Two figures in mirror symmetry
- Balanced on a tightrope (the hylo)
- Art deco / Afrofuturist aesthetic
- "Whether you're high or low" (Janelle Mon  e, Tightrope)

Consider using as:

- App logo/favicon
- Loading animation (figures dancing)
- Section dividers
- Easter egg on the About page

If the album "Fix Point" ever ships, the guide and the album share the same visual language.