Hylograph Tour: Higher-Order Structures Page

Status: draft **Category**: plan **Created**: 2026-01-28 **Tags**: hylograph, hats, tour, demo

Overview

A page in the Hylograph tour demonstrating visualization of higher-order data structures. Position: approximately one-third through the tour, after basic concepts are established.

Core message: "Pass your data structure to Hylograph, get back an illuminating picture."

Page Structure

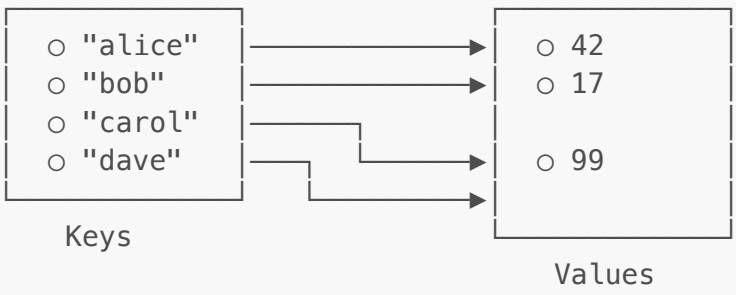
Header

Title: "Seeing Structure"

Tagline: "Maps, parsers, programs — visualized through a single abstraction."

Example 1: Data.Map — The Textbook Function Diagram

What it shows: A `Map k v` rendered as the classic domain-range-with-arrows diagram from mathematics textbooks.



Code snippet:

```
import Data.Map as Map
import Hylograph (visualize)

userAges :: Map String Int
userAges = Map.fromFoldable
  [ "alice" /\ 42
  , "bob" /\ 17
  , "carol" /\ 99
  , "dave" /\ 99
  ]

-- One line to visualize
visualize mapDiagram userAges
```

Why it matters:

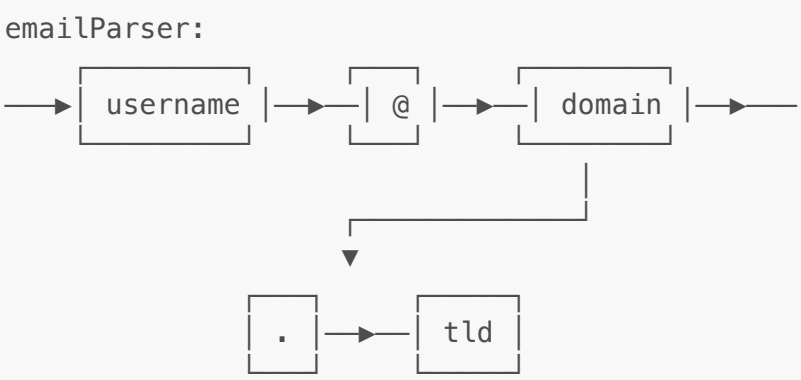
- Instantly recognizable representation
- Shows value sharing (carol and dave both → 99)
- Lowers friction to using Maps vs arrays
- Circle packing handles layout automatically

Technical notes:

- Domain and range extracted, then circle-packed separately
- Hylo assembles the positioned elements into HATS
- Links rendered as curves to avoid overlap

Example 2: Parser Combinators — Railroad Diagrams

What it shows: A parser definition transformed into a railroad/syntax diagram.



Code snippet:

```
import Parsing.Combinators
import Hylograph (visualize)

emailParser :: Parser String Email
emailParser = do
  user <- username
  _ <- char '@'
  dom <- domain
  _ <- char '.'
  tld <- topLevelDomain
  pure { user, dom, tld }

-- The parser IS the spec for the diagram
visualize railroadDiagram emailParser
```

Why it matters:

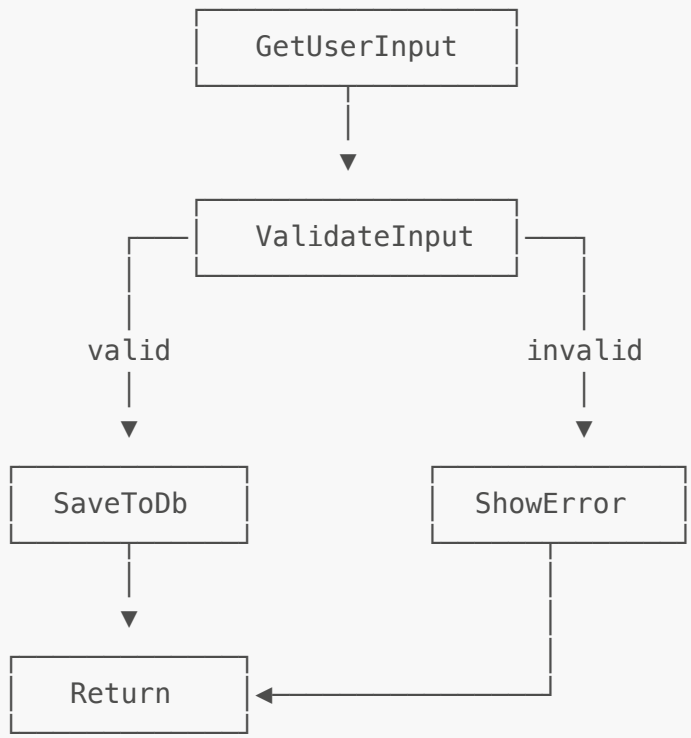
- Everyone recognizes railroad diagrams
- Almost nobody generates them from actual parsers
- Parser combinator structure (Alternative, sequence) maps directly to railroad semantics
- Documentation stays in sync with implementation

Technical notes:

- Parser combinators form a tree (sequence = chain, Alternative = branch)
- Hylo walks the combinator structure
- Algebra produces HATS nodes for each combinator type
- Choice (`<|>`) becomes parallel tracks
- Sequence (`<*>`, `<*`, `do`) becomes serial connection

Example 3: Free Monad Programs — Flowcharts

What it shows: A `Free f a` program rendered as a flowchart.



Code snippet:

```
import Control.Monad.Free
import Hylograph (visualize)

data AppF a
  = GetUserInput (String -> a)
  | ValidateInput String (Boolean -> a)
  | SaveToDb Record (Unit -> a)
  | ShowError String a
```

```
type App = Free AppF

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

-- Visualize the program structure
visualize flowchart myProgram
```

Why it matters:

- Free monads are "ASTs of your DSL" — this makes that literal
- Program structure becomes auditable/reviewable
- Great for explaining effects to newcomers
- Useful for debugging complex workflows

Technical notes:

- `Free f a` is literally a tree — perfect for hylo
- Each constructor of `f` becomes a node
- Continuations become edges
- Branching (`if`, pattern matching) shown explicitly
- Terminal `Pure` value shown as return node

Footer / Transition

Closing thought: "Same abstraction, three domains. The hylo doesn't care if it's folding a map, a parser, or a program — it sees trees all the way down."

Lead to next page: Teaser about custom interpreters — "What if you wanted Mermaid output instead of SVG? Or an English description for accessibility?"

Implementation Priority

1. **Map diagram** — Simplest, good starter, validates the pipeline
2. **Railroad diagram** — Highest "wow" factor, solves real problem
3. **Free flowchart** — Most technically impressive, but needs a good Free DSL to demo

Open Questions

- Which parser library to use for the demo? (Parsing from registry? Custom minimal one?)
- Should the Free example use a realistic DSL or a pedagogical one?
- Interactive elements? (hover to highlight corresponding code/diagram?)
- Do we show the HATS intermediate representation, or is that for a later page?

Related Documents

- `kb/architecture/hats-existential-design.md` — HATS architecture
- `kb/howto/hats-halogen-integration.md` — Integration patterns
- Tour page 1 (basics) — prerequisite content