Hylograph: Typed Visualization Through Structure-Transforming Folds

Abstract

D3.js introduced the "data join"—binding arrays to DOM elements via enter, update, and exit selections. The term is misleading; the operation is a fold. More precisely, it is a hylomorphism: an unfold of input structure followed by a fold into output structure. D3's version conflates four concerns: iteration, element creation, DOM structure, and differential updates. We decompose the fold into its categorical constituents: an enumeration (coalgebra) that unfolds input structure, an assembly (algebra) that folds into output structure, and a template that maps individual data to visual specification. Any input shape composes with any output shape. Type parameters enforce consistency between data, accessors, and visual encoding. We implement this design in PureScript as a library suitable for building reusable visualization components.

1. Introduction

Data visualization libraries occupy a spectrum. At one end, low-level toolkits like D3 offer power at the cost of complexity. At the other, high-level grammars like Vega-Lite trade expressiveness for ease. Both struggle with reusable components: D3's imperative style resists composition; declarative grammars resist extension.

The root issue is D3's core primitive. D3 calls it a "join," but it is not a join in any formal sense—not a SQL join, not a set-theoretic join, not a categorical join. It is a fold: a traversal that accumulates structure. Specifically, it is a hylomorphism—an unfold followed by a fold. D3's implementation binds an array to a selection, producing elements that mirror the array's structure. This works for flat collections but requires special handling for trees, graphs, and nested data. The abstraction leaks because the hylomorphism is implicit and constrained.

2. The Fold

A visualization transforms input structure into output structure. We model this as a hylomorphism—an unfold followed by a fold—with potentially different functors on each side.

Enumeration (coalgebra): How to traverse the input. An array yields elements sequentially. A tree yields elements in depth-first or breadth-first order. A graph yields nodes, edges, or spanning trees.

Template (natural transformation): How to visualize one datum. Maps a value to a visual specification independent of its position in the input or output structure.

Assembly (algebra): How to structure the output. Siblings produces flat DOM. Nested preserves hierarchy. GroupedBy partitions by key.

These three choices are orthogonal. Any enumeration composes with any assembly. The combinations form an N×M matrix from a single primitive.

We call this primitive a **fold**. When precision matters, we call it a **hylomorphic fold** to distinguish it from the simpler folds that reduce structure to a single value.

3. Type Safety Through Parametricity

Visualization components fail at runtime when accessors don't match data. A bar chart expects `d.value` but receives `d.amount`. JavaScript catches this only when the code runs. TypeScript catches some cases but its generics leak through escape hatches.

PureScript's type parameters are sound. A component signature:

```
barChart :: forall d. (d -> Number) -> (d -> String) -> Array d -> Tree d
```

The `forall d` means the component cannot inspect the data's structure—it can only apply the provided accessors. If the accessor types align with the data type, the component works. The compiler verifies this statically.

This enables genuine reuse. One bar chart implementation serves all data shapes. No runtime checks, no configuration objects, no accessor strings.

4. Implications for Component Libraries

D3's ecosystem shares code through examples. Users copy, paste, and modify. Abstraction is difficult because the imperative style couples data shape to implementation.

A typed fold as foundation changes what components can be:

- **Polymorphic by construction**: Components abstract over data shape via type parameters.
- **Composable**: The AST structure permits nesting, transformation, and inspection.
- **Verifiable**: The type checker validates composition before execution.

A component library built on this foundation would offer templates, enumerations, and assemblies as separate, combinable pieces rather than monolithic chart types.

5. Limitations and Future Work

The approach requires a language with sound parametric polymorphism. Adoption depends on tooling that lowers the barrier to working in such languages. Code generation, whether by humans or machines, may shift this calculus.

The fold handles structure transformation but does not address layout algorithms, interaction, or animation. These remain separate concerns, implementable atop the core abstraction.

6. Conclusion

D3's "data join" is a fold—a hylomorphism that unfolds input structure and folds it into output structure. Making this explicit separates enumeration from assembly, enabling composition across input and output shapes. Type parameters enforce accessor consistency without runtime checks. The result is a foundation for visualization components that are polymorphic, composable, and verifiable by construction.

Status / Next Steps

- Outline complete; ready for expansion into full paper
- Sections 3-4 could benefit from concrete code examples
- Consider adding section on AST representation and interpreter architecture
- Consider adding section on LLM-assisted composition as future work