Deriving Diagrams from Haskell/PureScript Types: The Catana Connection

Status: active **Category**: research **Created**: 2026-01-30 **Tags**: catana, recursion-schemes, hats, type-directed, meta-visualization

Summary

The Catana paper ("Folding over Neural Networks", MPC 2022) represents neural networks as recursive data types using Fix/Free. Since the network structure IS the type, diagrams of that structure could be derived automatically - a catamorphism over the type produces a HATS tree which renders to SVG.

This is meta: using hylomorphisms to visualize hylomorphisms.

The Catana Types

From the Catana Haskell implementation, neural networks are composed using free monads and coproducts "à la carte":

```
-- Layer types as base functors
data DenseF a = DenseF Int Int a         -- input_dim, output_dim, next
data ConvF a = ConvF KernelSize Stride a
data PoolF a = PoolF PoolSize a
data OutputF a = OutputF Int             -- no recursion - terminal

-- Network as fixed point
type Network = Fix (DenseF :+: ConvF :+: PoolF :+: OutputF)
```

The recursive structure of the type IS the diagram data.

The Derivation

A catamorphism over the network type produces a HATS tree:

```
-- Algebra: one layer of network → one layer of HATS
networkToHATS :: NetworkF HATSTree -> HATSTree
networkToHATS = case _ of
  DenseF inDim outDim next ->
    node "dense"
      [ attr "label" (show inDim <> " → " <> show outDim)
      , child next
      ]
  ConvF kernel stride next ->
    node "conv"
      [ attr "kernel" (show kernel)
      , attr "stride" (show stride)
      , child next
      ]
  PoolF size next ->
    node "pool"
      [ attr "size" (show size)
      , child next
      ]
  OutputF classes ->
    node "output"
      [ attr "classes" (show classes) ]

-- The derivation is just cata
derive :: Network -> HATSTree
derive = cata networkToHATS
```

No manual diagram construction - the structure emerges from the types.

Why This Matters

1. **Single source of truth**: Network definition IS diagram specification
2. **Always in sync**: Change the network, diagram updates automatically
3. **Type-safe**: Can't draw an invalid network topology
4. **Composable**: Coproducts of layer types = coproducts of diagram elements
5. **Meta-demonstration**: Hylograph visualizing hylomorphic computation

Connection to Catana's Architecture

The Catana paper shows that:

- Forward propagation = catamorphism (structure → value)
- Back propagation = anamorphism (seed → structure)
- Training = metamorphism (fold then unfold)

Our diagram derivation adds:

- Visualization = catamorphism (structure → HATS tree → DOM)

So the same conceptual machinery (recursion schemes over recursive types) handles both computation AND visualization.

Implementation Path

Phase 1: PureScript Port of Catana Types

```
-- In Zoo/NeuralTypes.purs
module Zoo.NeuralTypes where

data DenseF a = DenseF Int Int a
data ConvF a = ConvF { kernel :: Int, stride :: Int } a
data PoolF a = PoolF Int a
data OutputF a = OutputF Int

derive instance functorDenseF :: Functor DenseF
-- etc.

-- Coproduct for composition
data (:+:) f g a = InL (f a) | InR (g a)
infixr 6 type (:+:) as :+:
```

Phase 2: HATS Algebra

```
-- In Zoo/NeuralDiagram.purs
module Zoo.NeuralDiagram where

import Zoo.NeuralTypes
import HATS (HATSTree, node, attr, child)

class ToHATS f where
  toHATS :: f HATSTree -> HATSTree

instance toHATSDense :: ToHATS DenseF where
  toHATS (DenseF inD outD next) =
    node "g"
      [ attr "class" "layer dense"
      , child (rect inD outD)  -- visual representation
      , child next
      ]

-- Coproduct instance lifts automatically
instance toHATSCoproduct :: (ToHATS f, ToHATS g) => ToHATS (f :+: g) where
  toHATS (InL fa) = toHATS fa
  toHATS (InR ga) = toHATS ga
```

Phase 3: Layout

Neural network diagrams typically use layered/Sugiyama layout. The derived HATS tree feeds into the layout algorithm:

```
Network type → cata toHATS → HATS tree → Sugiyama layout → positioned HATS
→ D3 render
```

This is where the planned Sugiyama layout work connects.

Rendering Options

The same derived HATS tree could render as:

1. **Layered diagram** (like Catana paper figures 2, 4)

2. **Nested boxes** (showing dimension transformations)
3. **Flow diagram** (forward/backward arrows)
4. **English description** ("A network with 3 dense layers...")
5. **Mermaid syntax** (for documentation)

Different HATS interpreters, same source type.

Interactive Extensions

Since we have the full type structure:

- **Click layer**: Show parameters, activations
- **Hover**: Highlight data flow path
- **Animate training**: Metamorphism in action (fold up, unfold down)
- **Compare architectures**: Side-by-side type diffs → visual diffs

Connection to Morphism Zoo

The psd3-prim-zoo-mosh showcase already has the recursion scheme infrastructure. Adding neural network types would:

1. Provide a new domain for demonstrating schemes
2. Connect to the Catana paper directly
3. Show that the "zoo" isn't just pedagogical - it's a real visualization technique

Open Questions

1. **How to handle coproducts visually?** Different layer types need distinct visual treatment while maintaining structural consistency.

2. **Dimension annotations**: Neural networks have rich dimensional metadata (input/output shapes, kernel sizes). How to show without clutter?

3. **Training animation**: Could we visualize the metamorphism (forward → backward) as an animation over the same structure?

4. **Type-level vs value-level**: Catana uses dependent-ish types for dimension checking. Can we reflect that in the visualization?

References

- Catana repo
- Paper: Folding over Neural Networks (arXiv)
- ar5iv HTML version
- MPC 2022 proceedings (Springer)

Related

- Recursion Schemes as Visualization Primitives
- L-Systems Visualization
- Sugiyama Layered Graph Layout