SSRS Integration for HATS

Summary

Spike exploring whether purescript-ssrs can be adopted for HATS (Hylomorphic Abstract Tree Syntax). The library provides stack-safe recursion schemes via Conor McBride's dissection technique.

Verdict: Partially adopt. Use ssrs for the *interpretation* side (HATS → DOM is a cata). Keep input side (data → HATS) as regular functions for now.

What SSRS Provides

```
purescript-fixed-points     Mu (least fixpoint)
         ↓
purescript-dissect          Dissect typeclass (stack-safe traversal)
         ↓
purescript-ssrs             cata, ana, hylo, para, histo, zygo, apo,
futu...
```

Key types:

```
newtype Mu f = In (f (Mu f))           -- Recursive type
type Algebra p v = p v -> v             -- Fold
type Coalgebra p v = v -> p v           -- Unfold
cata :: Dissect p q => Algebra p v -> Mu p -> v
hylo :: Dissect p q => Algebra p v -> Coalgebra p w -> w -> v
```

Spike Results

See `spikes/ssrs-hats-spike/` for working code.

What Works Perfectly

1. **Cata with flattening**: Tree structure → flat array

   ```
   flattenAlgebra :: Algebra RoseF (Array Circle)
   flattenAlgebra (RoseF label children) =
     [ { label, depth: 0 } ] <> concat children

   cata flattenAlgebra tree  -- Tree → [circle, circle, ...]
   ```

2. **Cata with nesting**: Same tree → nested DOM

```
nestAlgebra :: Algebra RoseF DOMNode
nestAlgebra (RoseF label children) =
  case children of
    [] -> Circle label
    _  -> Element "g" (cons (Circle label) children)
```

3. **Hylo with fusion**: Unfold + fold with NO intermediate materialization

```
generateAndFlatten = hylo flattenAlgebra seedCoalgebra
-- Intermediate tree is VIRTUAL
```

4. **Accumulating algebra**: Thread context (depth, parent, etc.)

```
-- Result type is a FUNCTION that receives context
withDepth :: Algebra RoseF (Int -> Array Circle)
withDepth (RoseF label childFns) = \depth ->
  [ { label, depth } ] <> concat (map (\f -> f (depth+1)) childFns)
```

The Limitation

Hylo requires the SAME pattern functor on both sides.

```
hylo :: Algebra p v -> Coalgebra p w -> w -> v
--              ^               ^
--              Same p!
```

Can't directly do `InputF input → OutputF output` if InputF ≠ OutputF.

Solutions for Heterogeneous Transformations

1. **Accumulating algebra** - Thread extra state via function type
2. **Natural transformation** - If `InputF ≅ OutputF`, use `transMu`
3. **Two-phase** - `cata input → value → ana output` (materializes intermediate)
4. **À la carte** - Coproduct functor `(InputF :+: OutputF)` encompasses both
5. **Make intermediate flexible** - If HATS IS the functor, it must represent all shapes

Implications for HATS

Current HATS Structure

```
data Tree
  = Elem { elemType, attrs, children :: Array Tree, behaviors }
```

```
  | MkFold SomeFold  -- Existentially-wrapped iteration
  | Empty
```

With SSRS (Pattern Functor)

```
data TreeF a
  = ElemF ElementType (Array Attr) (Array a) (Array ThunkedBehavior)
  | FoldF SomeFold
  | EmptyF

type Tree = Mu TreeF

-- Interpreter becomes an algebra
interpretToD3 :: Algebra TreeF (Effect Unit)
```

The Awkwardness: MkFold / FoldF

The existential `SomeFold` inside the functor is awkward because:

- It's not itself a recursive position
- It contains its own enumeration (coalgebra-like)
- It breaks the clean functor structure

Options:

1. **Remove FoldF** - Always expand folds before creating Tree
2. **Two-level** - Outer Tree, inner Fold, interpret separately
3. **Accept hybrid** - Use ssrs for Elem nodes, handle Fold specially

Recommendation

Phase 1: Adopt ssrs for interpretation

- Make `TreeF` a proper pattern functor (without FoldF initially)
- `cata interpretToD3 :: Mu TreeF -> Effect Unit`
- Stack-safe, principled, matches the "Hylograph" name

Phase 2: Handle Fold separately

- `forEach` builds `Mu TreeF` directly (expand during construction)
- Or: interpret Fold as a separate pass before main cata

Defer: Full hylo

- Requires making "build HATS from data" a proper coalgebra
- May be overkill for most use cases
- Keep as regular functions unless bottleneck appears

Dissect Instance for HATSF

Will need a `Dissect TreeF TreeQ` instance. The dissect library supports generic deriving for simple cases, but HATSF's array children may need manual implementation.

Example from spike:

```
data RoseQ c j = RoseQ String (Array c) (Array j)

instance Dissect RoseF RoseQ where
  init (RoseF label children) = case uncons children of
    Nothing -> return (RoseF label [])
    Just { head, tail } -> yield head (RoseQ label [] tail)

  next (RoseQ label clowns jokers) val = case uncons jokers of
    Nothing -> return (RoseF label (snoc clowns val))
    Just { head, tail } -> yield head (RoseQ label (snoc clowns val) tail)
```

References

- purescript-ssrs - Stack-safe recursion schemes
- purescript-dissect - Dissection typeclass
- Clowns to the Left of me, Jokers to the Right - McBride's paper
- Tim Williams's recursion-schemes slides - Derivations of schemes
- Spike: `spikes/ssrs-hats-spike/`