Indexed + Tagless Final: Feasibility Exploration

Question: Can we combine indexed monads with tagless final style for extensible, type-safe interpreters?

The Challenge

We want:

1. **Indexed monad safety** - State transitions tracked in types (SPending → SBound)
2. **Tagless final extensibility** - Multiple interpreters (D3v2, String, Meta, Music)
3. **Clean user code** - No complex type signatures in examples

Attempt 1: Indexed Type Class

```
-- Type class with indexed monad constraint
class IxMonad m <= IxSelectionM m where
  select
    :: String
    -> m SEmpty SEmpty (Selection SEmpty Element Unit)

  append
    :: forall parent datum
     . ElementType
    -> Array (Attribute datum)
    -> Selection SPending parent datum
    -> m SPending SBound (Selection SBound Element datum)

  setAttrs
    :: forall datum
     . Array (Attribute datum)
    -> Selection SBound Element datum
    -> m SBound SBound (Selection SBound Element datum)

  remove
    :: forall datum
     . Selection SExiting Element datum
    -> m SExiting SExiting Unit
```

User code:

```
drawCircles :: forall m. IxSelectionM m => m SEmpty SBound Unit
drawCircles = ado  -- or Ix.do
  svg <- select "svg"
  JoinResult { enter, update, exit } <- joinData [1, 2, 3] "circle" svg
  enterEls <- append Circle [...] enter
  updateEls <- setAttrs [...] update
  _ <- remove exit
  in unit
```

Problem 1: User signatures are complex

- User must specify `m SEmpty SBound Unit`
- Need to know the final state of their computation
- Loses the simplicity of `forall m. SelectionM sel m => m Unit`

Attempt 2: Existential Output State

```
-- Hide the output state
drawCircles :: forall m o. IxSelectionM m => m SEmpty o Unit
drawCircles = ado
  svg <- select "svg"
  ...
```

Problem 2: Can't compose

- If one function ends in state `o`, how does the next function know what state it starts in?
- This breaks composition

Attempt 3: Index-Erased User API

```
-- Type class that USES indexed monad internally but exposes simple API
class Monad m <= SelectionM sel m | m -> sel where
  renderData
    :: forall f datum
     . Foldable f
    => Ord datum
    => ElementType
    -> f datum
    -> String
    -> sel  -- No state parameter!
    -> Maybe (datum -> Array (Attribute datum))
    -> Maybe (datum -> Array (Attribute datum))
    -> Maybe (datum -> Array (Attribute datum))
    -> m sel  -- Returns unindexed selection

-- Interpreter implementation
newtype D3v2M a = D3v2M (Effect a)

instance SelectionM D3Selection_ D3v2M where
  renderData elemType data selector sel enterAttrs updateAttrs exitAttrs =
D3v2M do
    -- Internally uses indexed monad for safety
    runIxSelectionM $ do
      empty <- coerceToEmpty sel
      JoinResult { enter, update, exit } <- joinData data selector empty
      ... indexed operations ...
    -- Returns in regular Effect monad
```

Problem 3: Loses the point

- If we erase indices at the API boundary, we lose compile-time state tracking
- User can still make sequencing errors in their code
- The indexed monad only helps library internals

Attempt 4: Interpreter-Specific State Tracking

What if different interpreters track different things?

```
-- Generic indexed constraint
class IxMonad m <= IxSelectionM m where
  select :: String -> m i i (Selection ...)  -- i is polymorphic!
  append :: Selection SPending ... -> m j k (Selection SBound ...)
```

Problem 4: State polymorphism is too loose

- If `i` is polymorphic, it means "any state", defeating the purpose
- We need concrete states (SPending, SBound) for safety

The Fundamental Tension

Indexed monads require:

- Input state `i` and output state `o` in the type
- Concrete states for safety (SPending, not `forall i`)
- State must be known at compile time

Tagless final requires:

- Polymorphism over the monad `m`
- User code shouldn't mention implementation details
- Different interpreters can have different semantics

The conflict:

- D3v2 interpreter: **Needs** state tracking (DOM state is real)
- String interpreter: **Doesn't need** state tracking (just building strings)
- Meta interpreter: **Might need** different state tracking (AST nodes)
- Music interpreter: **Definitely different** states (audio synthesis)

What Actually Works

Option A: Indexed Monad for Direct Use (No Tagless Final)

```
-- Users import PSD3v2.Indexed directly
import PSD3v2.Indexed as PSD3
```

```
import Control.Monad.Indexed as Ix

drawCircles :: IxSelectionM SEmpty SBound Unit
drawCircles = Ix.do
  svg <- select "svg"
  JoinResult { enter } <- joinData [1, 2, 3] "circle" svg
  circles <- append Circle [...] enter
  pure unit

-- Run it
main = runIxSelectionM drawCircles
```

Consequences:

- ✅ Full type safety with state tracking
- ✅ Clear error messages ("can't append to SEmpty")
- ✅ Direct, no interpreter indirection
- ❌ Not extensible - only one "interpreter" (the DOM one)
- ❌ Can't generate HTML strings or AST
- ❌ User signatures include state indices

Option B: Tagless Final (No Indexed Monad)

```
-- Current approach
drawCircles :: forall m sel. SelectionM sel m => m Unit
drawCircles = do
  svg <- select "svg"
  circles <- renderData Circle [1, 2, 3] "circle" svg ...
  pure unit

-- Multiple interpreters
eval_D3v2 :: D3v2M ~> Effect
eval_String :: StringM ~> Identity
eval_Meta :: MetaM ~> Writer (Array MetaOp)
```

Consequences:

- ✅ Fully extensible
- ✅ Multiple interpreters
- ✅ Clean user signatures
- ⚠ Runtime state checking (via unsafePartial in implementation)
- ⚠ Phantom types provide *some* compile-time safety

Option C: Hybrid - Indexed in D3v2, Not in Others

```
-- Type class uses regular monad
class Monad m <= SelectionM sel m | m -> sel where
  renderData :: ... -> m sel
```

```
-- D3v2 interpreter uses indexed monad INTERNALLY
newtype D3v2M a = D3v2M (Effect a)

instance SelectionM D3Selection_ D3v2M where
  renderData elemType data selector sel enterAttrs updateAttrs exitAttrs =
D3v2M do
    -- Implementation uses IxSelectionM for safety
    runIxSelectionM internalIndexedVersion
    where
      internalIndexedVersion :: IxSelectionM SEmpty SBound (Selection
SBound ...)
      internalIndexedVersion = Ix.do
        ... type-safe indexed operations ...

-- String interpreter doesn't need indexed monad
instance SelectionM StringSelection StringM where
  renderData = ... just build HTML string ...
```

Consequences:

- ✅ Extensible (multiple interpreters)
- ✅ D3v2 implementation is type-safe internally
- ✅ Clean user signatures
- ⚠ User code doesn't get state tracking
- ⚠ User could write `renderData` with wrong sequencing, caught at D3v2 runtime

Verdict

It is NOT possible to have both indexed monad user-facing safety AND tagless final extensibility simultaneously.

The fundamental issue: Indexed monads require the user's type signature to include state indices (`m i o a`), but tagless final requires the user's type signature to be polymorphic over the monad (`m a`). These are incompatible.

Recommendation

Choose based on priority:

Priority: Extensibility → Use Tagless Final (Option B)

- Multiple interpreters (D3v2, String, Meta, Music)
- Clean user code: `forall m sel. SelectionM sel m => m Unit`
- Safety via phantom types + unsafePartial (hidden in library)
- This is what we have in current PSD3

Priority: Type Safety → Use Indexed Monad (Option A)

- State tracking in user code
- Compiler prevents sequencing errors

- User signatures: `IxSelectionM SEmpty SBound Unit`
- Only one "interpreter" (DOM operations)
- Good for library users who want maximum safety

Hybrid: Both APIs (Recommended)

Provide BOTH:

1. **PSD3v2.Indexed** - Direct indexed monad API
   - For users who want maximum compile-time safety
   - For library code / advanced users
   - Explicit state tracking
2. **PSD3v2 with Tagless Final** - Extensible interpreter API
   - For examples and typical usage
   - Clean signatures
   - Multiple interpreters
   - Uses Indexed internally for D3v2 implementation safety

User chooses based on needs:

```
-- Simple example (tagless final)
import PSD3 as PSD3

drawCircles :: forall m sel. SelectionM sel m => m Unit
drawCircles = PSD3.do
  svg <- select "svg"
  ...

-- Maximum safety (indexed monad)
import PSD3v2.Indexed as Ix
import Control.Monad.Indexed (discard) as Ix

drawCircles :: IxSelectionM SEmpty SBound Unit
drawCircles = Ix.do
  svg <- select "svg"
  ...
```

Final Answer

No, you cannot have indexed monad type-level state tracking in user code while maintaining tagless final extensibility.

The indexed monad approach and tagless final approach solve different problems:

- **Indexed monad**: Prevents sequencing errors at compile time (single semantics)
- **Tagless final**: Enables multiple interpretations (extensibility)

You must choose which property is more important, or provide both APIs and let users choose.

For the PSD3 project, I recommend **tagless final** as primary (extensibility is core to the vision), with indexed monad as an optional advanced API.