A Language for Describing Optimization Strategies

1. Wouldn’t it be great...
   - to look behind the curtains of optimizing compilers and actually understand how optimizations are applied
   - to have a flexible way of specifying optimizations for your rewrite-based compiler
   - to build custom optimizations in an extensible language while avoiding to rely on fixed scheduling APIs
   - to use a scalable approach that hides complexity behind high-level abstractions

2. Core Concepts
   - A Strategy encodes a program transformation
     type Strategy[P] = P → RewriteResult[P]
   - A RewriteResult encodes its success or failure
     RewriteResult[P] = Success(p: P) | Failure(p: Strategy[P])
   - Examples
     - def id[P]: Strategy[P] = (p: P) → Success(p)
     - def fail[P]: Strategy[P] = (p: P) → Failure(fail)
     - ...and language-specific: map(f) = map(g) → map (f ∘ g)
   - def mapFusion: Strategy[Lift] = p → match
     case map(f) = map(g) → Success(map(f ∘ g))
     case _ → Failure(mapFusion)

3. Case Studies
   - Automatic Differentiation
     - Efficient Differentiable Programming in a Functional Array-Processing Language
     - F achieves efficiency by rewriting differentiated code
   - Image Processing
     - Traversals: Where to apply a strategy?
       $\lambda f. \text{map } f \rightarrow f \text{ apply map } f$
     - Traversing AST:
       - all:
         - one:
           - some:
     - Combinators
       - Choice
         - try
         - repeat
     - Image Processing:
       - $[1 \ 0 \ 1] \times [-1 \ 0 \ 1]$
     - Not expressible as a schedule in Halide! requires modification of the algorithm instead
   - Deep Learning
     - Implementing TVM’s scheduling language:
       - Halide:
         - Separable Convolution: Sobel Filter
       - Not expressible as a schedule in Halide! requires modification of the algorithm instead
       - Val blocking:
         - (topdown(tile(32,22)))
         - (topdown(isReduce; split(4)))
         - (topdown(reorder(Seq(1,2,5,6,3,4)) ; blocking ; lowerFC)(nn))
     - Extensible: ELEVATE can be extended with custom domain-specific optimizations
     - Scalable: ELEVATE hides 100K’s of rewrite steps behind high-level abstractions