Overview

1. Who We Are
2. Getting FP in the Door
   Monoids, Reducers, Iteratees, Performance Attribution
3. Ermine
   “Haskell with Row Types”
4. Reporting
   EDSL Inception and JMacro RPC
5. Open Source
Overview

1. Who We Are
2. Getting FP in the Door
   Monoids, Reducers, Iteratees, Performance Attribution
3. Ermine
   "Haskell with Row Types"
4. Reporting
   EDSL Inception and JMacro RPC
5. Open Source
Who We Are

Over 4,500 customers including

- Investment Management Firms
- Private Equity Funds
- Investment Banks
- Advisory Firms
- Corporations
- Universities
Who We Are

Products Include

- S&P Capital IQ Platform
- ClariFI
- AlphaWorks
- Compustat
Overview

1. Who We Are

2. Getting FP in the Door
   Monoids, Reducers, Iteratees, Performance Attribution

3. Ermine
   “Haskell with Row Types”

4. Reporting
   EDSL Inception and JMacro RPC

5. Open Source
Getting FP in the Door

- Monoids
- Reducers
- Performance Attribution
- Factor Backtesting
Monoids

class Monoid m where
  mappend :: m -> m -> m
  mempty :: m

newtype Sum a = Sum { getSum :: a }
instance Num a => Monoid (Sum a) where
  mempty = Sum 0
  mappend (Sum m) (Sum n) = Sum (m + n)

This is really all written in Scala!
Monoids

class Monoid m where
    mappend :: m -> m -> m
    mempty :: m

>>> foldMap Sum [1,2,3]
6

This is really all written in Scala!
Reducers

class Monoid m where
    mappend :: m -> m -> m
    mempty :: m

data Reducer a b =
    forall m. Monoid m => R (a -> m) (m -> b)

reduce :: Reducer a b -> [a] -> b
reduce (R am mb) xs = mb (foldMap am xs)

In practice you probably want an efficient cons and/or snoc operation as well.
Simultaneous Reduction

instance (Monoid a, Monoid b) => Monoid (a, b)

instance Applicative (Reducer a) where
  pure a = R (\_ -> ()) (\() -> a)
  R am mf <*> R an nx = R (am &&& an) $
    \(m,n) -> mf m $ nx n

mean = (/) <$> sum <*> length

This really needs a strict pair.
Simultaneous Reduction

instance Num b => Num (Reducer a b)
instance Fractional b => Fractional (Reducer a b)

mean = sum / length
Performance Attribution

OLD AND BUSTED
- Implemented in Java
- Needed Full Dataset in Memory
- Hard to Extend
- Results are Serialized Objects

NEW HOTNESS
- Implemented in Scala
- Runs in Constant Memory
- Easily Extended
- Drastic Speed Improvements
- Results Flattened via Combinators to Database
Performance Attribution

OLD AND BUSTED
- Implemented in Java
- Needed Full Dataset in Memory
- Hard to Extend
- Results are Serialized Objects

NEW HOTNESS
- Implemented in Scala
- Runs in Constant Memory
- Easily Extended
- Drastic Speed Improvements
- Results Flattened via Combinators to Database

GLOSSED OVER
- Multi-Pass Algorithms
- Iteratees
- Caching
- Aggregation by Sectors
- Missing Data
Portfolio Attribution Report

Summary report for long/short portfolios

This report has been exported from a performance attribution workflow run in ModelStation from Clarifi

**REPORT INFORMATION**

General information about the Portfolio Attribution Report run in ModelStation and exported to create this workbook.

<table>
<thead>
<tr>
<th>Portfolio Name</th>
<th>Dow Long / Short</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benchmark</td>
<td>4 stock</td>
</tr>
<tr>
<td>Returns Attribution</td>
<td>Beginning of period holdings</td>
</tr>
<tr>
<td>Risk Model</td>
<td>Capital IQ Risk Model</td>
</tr>
<tr>
<td>Analysis Period</td>
<td>110/11/30 – 111/2/30</td>
</tr>
</tbody>
</table>

**RETURNS**

These statistics summarize the characteristics of the returns (losses) of the portfolio over the duration of the analysis period. ModelStation analysis, benchmark-relative statistics are also provided for the period.

**BRINSON ATTRIBUTION**

These charts and tables show the cumulative attribution of the Brinson effects. Periodic data is linked using Carino smoothing.
Overview

1. Who We Are
2. Getting FP in the Door
   Monoids, Reducers, Iteratees, Performance Attribution
3. Ermine
   “Haskell with Row Types”
4. Reporting
   EDSL Inception and JMacro RPC
5. Open Source
Ermine
Ermine
“Haskell with Row Types”

FEATURES
• Not Scala
• Portable Core that can run on the JVM
• Implementations in Scala and Haskell
• Strong Type System
  • Novel Row Types
  • Polymorphic and Constraint Kinds
  • Rank-N Types via a derivative of HMF
• Fits our Domain
  • Built-In Database Support
  • Can Be Exposed to End Users
  • Prototype Structured Code Editor
  • Full Control Over Error Messages

It actually has many differences with Haskell.
PREVIOUS APPROACHES

Commonly row polymorphism is modeled via has, lacks, and/or subsumes constraints.

\[
\begin{align*}
\text{cons}_1 & : \forall (a:*)(r:\rho). (r/1) \Rightarrow \alpha \rightarrow [..r] \rightarrow \{1:\alpha| r\} \\
\text{tail}_1 & : \forall (\alpha:*)(r:\rho). (r/1) \Rightarrow \{1:\alpha| r\} \rightarrow \{r\} \\
\text{join} & : \forall (r:\rho)(s:\rho). [..r] \rightarrow [..s] \rightarrow [..r||s]
\end{align*}
\]

This is easily checked, but now inference flows unidirectionally through join.
Row Polymorphism

**IN ERMINE**
We use a single constraint type: “can be partitioned into”

\[ a \leftarrow (b,c) \]

says the fields in row type a can be partitioned into disjoint sets of fields b and c.

join:
\[
(\begin{array}{l}
\text{d} \leftarrow (a,b) \\
, \text{e} \leftarrow (b,c) \\
, \text{f} \leftarrow (a,b,c)
\end{array}) \Rightarrow [..d] \rightarrow [..e] \rightarrow [..f] \]
IN ERMINI
We can have existentials in our constraint types. E.g.

```haskell
forget : MonadState s m => m a -> m a
forget m = do s ← get; a ← m; put s; return a
```

has type parameters that don’t occur on the right hand side of the \(\Rightarrow\) determined by functional dependencies. We could give this type:

```haskell
forget : (\exists s. MonadState s m) \Rightarrow m a \to m a
```

Using this we can model “Has” and “Lacks” or “Disjoint” via type aliases!

```haskell
type Has a b = \exists c. a \leftarrow (b, c)
type a | b = \exists c. c \leftarrow (a, b)
```
Overview

1. Who We Are
2. Getting FP in the Door
   Monoids, Reducers, Iteratees, Performance Attribution
3. Ermine
   “Haskell with Row Types”
4. Reporting
   EDSL Inception and JMacro RPC
5. Open Source
Reporting
A Language in our Language

 portfolio Information

<table>
<thead>
<tr>
<th>Portfolio Name</th>
<th>big portfolio</th>
<th>Date Range</th>
<th>4/30/2012 to 4/30/2013</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portfolio ID</td>
<td>-</td>
<td>Currency</td>
<td>US Dollar (USD)</td>
</tr>
<tr>
<td>Benchmark</td>
<td>Russell Top 50 Index (&quot;RTF&quot;)</td>
<td>Grouping</td>
<td>GICS Industry</td>
</tr>
<tr>
<td>Risk Model</td>
<td>Global Fundamental Short Term</td>
<td>Exclude Cash</td>
<td>No</td>
</tr>
</tbody>
</table>

portfolio overview

<table>
<thead>
<tr>
<th>Port</th>
<th>Bench</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Return</td>
<td>28.98</td>
<td>14.37</td>
</tr>
<tr>
<td>Realized Risk</td>
<td>16.29</td>
<td>12.28</td>
</tr>
<tr>
<td>Latest Predicted Risk</td>
<td>12.99</td>
<td>8.69</td>
</tr>
<tr>
<td>Factor Risk</td>
<td>9.79</td>
<td>8.33</td>
</tr>
<tr>
<td>-Stock Specific Risk</td>
<td>8.54</td>
<td>2.47</td>
</tr>
<tr>
<td>Portfolio Value($mm)</td>
<td>0.15</td>
<td>-</td>
</tr>
<tr>
<td># of Securities</td>
<td>17</td>
<td>52</td>
</tr>
<tr>
<td>Annualized Total Return</td>
<td>28.98</td>
<td>14.37</td>
</tr>
</tbody>
</table>

industry weight

<table>
<thead>
<tr>
<th>Port</th>
<th>Bench</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Software and Services</td>
<td>55.11</td>
<td>3.23</td>
</tr>
<tr>
<td>Automobiles</td>
<td>12.98</td>
<td>0.00</td>
</tr>
<tr>
<td>Capital Markets</td>
<td>11.40</td>
<td>0.00</td>
</tr>
<tr>
<td>Pharmaceuticals</td>
<td>5.70</td>
<td>10.54</td>
</tr>
<tr>
<td>Food Products</td>
<td>4.78</td>
<td>1.24</td>
</tr>
<tr>
<td>Diversified Financial Services</td>
<td>3.28</td>
<td>6.55</td>
</tr>
<tr>
<td>Commercial Services and Retail</td>
<td>1.51</td>
<td>0.00</td>
</tr>
<tr>
<td>Electric Utilities</td>
<td>1.50</td>
<td>1.50</td>
</tr>
<tr>
<td>Industrial Conglomerates</td>
<td>1.49</td>
<td>4.43</td>
</tr>
<tr>
<td>Trading Companies and Diversified</td>
<td>0.83</td>
<td>0.00</td>
</tr>
<tr>
<td>Biotechnology</td>
<td>0.71</td>
<td>1.16</td>
</tr>
<tr>
<td>Other</td>
<td>0.73</td>
<td>72.85</td>
</tr>
</tbody>
</table>

security weight

<table>
<thead>
<tr>
<th>Port</th>
<th>Bench</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Inc. (NasdaqGS:GOOG)</td>
<td>55.11</td>
<td>3.06</td>
</tr>
<tr>
<td>Volkswagen AG (DB:VOW)</td>
<td>12.98</td>
<td>0.00</td>
</tr>
<tr>
<td>The Goldman Sachs Group, Inc.</td>
<td>9.76</td>
<td>0.00</td>
</tr>
<tr>
<td>Johnson &amp; Johnson (NYSE:JNJ)</td>
<td>5.70</td>
<td>3.36</td>
</tr>
<tr>
<td>Nestl© S.A. (SWX: NESN)</td>
<td>4.78</td>
<td>0.00</td>
</tr>
<tr>
<td>JPMorgan Chase &amp; Co. (NYSE:JPM)</td>
<td>3.28</td>
<td>2.68</td>
</tr>
<tr>
<td>ABM Industries Incorporated</td>
<td>1.51</td>
<td>0.00</td>
</tr>
<tr>
<td>Electrotécnica de France SA (ELEB)</td>
<td>1.50</td>
<td>1.50</td>
</tr>
<tr>
<td>General Electric Company (GE)</td>
<td>1.49</td>
<td>3.39</td>
</tr>
<tr>
<td>Gluskin Sheff + Associates, Inc.</td>
<td>1.18</td>
<td>0.00</td>
</tr>
<tr>
<td>Total</td>
<td>97.28</td>
<td>12.49</td>
</tr>
</tbody>
</table>

security contribution

<table>
<thead>
<tr>
<th>Port</th>
<th>Avg Wgt</th>
<th>Return</th>
<th>Contr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Inc. (NasdaqGS:GOOG)</td>
<td>53.86</td>
<td>36.33</td>
<td>19.16</td>
</tr>
<tr>
<td>Volkswagen AG (DB:VOW)</td>
<td>14.17</td>
<td>16.33</td>
<td>2.78</td>
</tr>
<tr>
<td>The Goldman Sachs Group, Inc.</td>
<td>9.33</td>
<td>28.98</td>
<td>2.53</td>
</tr>
<tr>
<td>Johnson &amp; Johnson (NYSE:JNJ)</td>
<td>5.53</td>
<td>35.63</td>
<td>1.92</td>
</tr>
<tr>
<td>Nestlé S.A. (SWX: NESN)</td>
<td>5.01</td>
<td>20.34</td>
<td>1.02</td>
</tr>
</tbody>
</table>

bottom 5

<table>
<thead>
<tr>
<th>Port</th>
<th>Avg Wgt</th>
<th>Return</th>
<th>Contr</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABM Industries Incorporated</td>
<td>1.95</td>
<td>-0.25</td>
<td>-0.18</td>
</tr>
<tr>
<td>Sumitomo Corporation (TSE:8001)</td>
<td>1.02</td>
<td>-9.04</td>
<td>-0.14</td>
</tr>
<tr>
<td>Geofon SpA (BIT:GE)</td>
<td>0.28</td>
<td>-24.41</td>
<td>-0.12</td>
</tr>
<tr>
<td>Granville Pacific Capital Corporation</td>
<td>0.01</td>
<td>-3.91</td>
<td>0.00</td>
</tr>
<tr>
<td>Advanced Power Compon...</td>
<td>0.02</td>
<td>241.45</td>
<td>0.03</td>
</tr>
</tbody>
</table>
contributionSnapshotReport rid = dropdown
(remoteChoice dateRangePresentation (reportingRanges rid)) $
\dateRangeDropdown dateRangeSelector ->
topAndCenter
(hugL $ hflow ["Selected Period: ", dateRangeDropdown])
(using dateRangeSelector (\ dr -> topAndCenter
(contributionSnapshotSummary rid (dr ! dateRangeId))
(tabbed [
("Net", contributionSnapshotReportContent rid (dr ! dateRangeId) LSNNet),
("Long", contributionSnapshotReportContent rid (dr ! dateRangeId) LSNLong),
("Short", contributionSnapshotReportContent rid (dr ! dateRangeId) LSNShort)
]))
)
The same report specification can run under wildly different interpreters. E.g.

- HTML + AJAX
- Excel
- PDF
- Java Swing
Overview

1. Who We Are
2. Getting FP in the Door
   Monoids, Reducers, Iteratees, Performance Attribution
3. Ermine
   “Haskell with Row Types”
4. Reporting
   EDSL Inception and JMacro RPC
5. Open Source
Open Source

SCALA VERSION
- https://github.com/ermine-language/ermine-legacy

HASKELL VERSION
- https://github.com/ermine-language/ermine

NEW CORE INTERPRETER
- https://github.com/ermine-language/ermine-scala-core

JMACRO-RPC
Questions?