How Profilers Can Help Navigate Type Migration

* Ben Greenman
  Matthias Felleisen
  Christos Dimoulas

BYU PL Seminar
How Profilers Can Help Navigate Type Migration
How Profilers Can Help Navigate Type Migration

How to avoid runtime costs using off-the-shelf tools?
How Profilers Can Help Navigate Type Migration

How to avoid runtime costs using off-the-shelf tools?

 costs ~ gradual types
 tools ~ statistical profilers
Old Problem, New Idea
Old Problem, New Idea

popl'16: 10x slowdowns are common, but fast points exist!
Old Problem, New Idea

popl'16: 10x slowdowns are common, **but** fast points exist!

Is Sound Gradual Typing Dead?
Asama Takikawa, Daniel Feeley, Ben Greenman, Max S. New, Jan Vitek, Matthias Felleisen
Northeastern University, Boston, MA

Abstract

Some gradual typing systems are sound, others are not. This paper surveys many such systems and identifies common flaws. It shows how to fix them.

Rational Programmer method (icfp'21)

How to find??
Gradual Types + Costs
def avg(g):
    return mean(get_column(g, "score"))

def mean(nums):
    ....

def get_column(table, col_name):
    ....

avg(quiz_1_grades)
avg(recipe_book)
avg(42)
Gradual Types + Costs

avg : Gradebook -> Num
def avg(g):
    return mean(get_column(g, "score"))

def mean(nums):
    ....

def get_column(table, col_name):
    ....

avg(quiz_1_grades)
avg(recipe_book)
avg(42)

Add types, code still runs
avg : Gradebook -> Num
def avg(g):
    return mean(get_column(g, "score"))

avg(quiz_1_grades)
avg(recipe_book)
avg(42)
Gradual Types + Costs

```python
avg : Gradebook -> Num
def avg(g):
    return mean(get_column(g, "score"))
```

Type soundness needs Runtime checks

- `avg(quiz_1_grades)`
- `avg(recipe_book)`
- `avg(42)`
Gradual Types + Costs

avg : Gradebook -> Num
def avg(g):
    return mean(get_column(g, "score"))

avg(quiz_1_grades)
avg(recipe_book)
avg(42)

Type soundness needs Runtime checks

Costs depend ...

Guarded semantics  
**deep** types
Contract @ boundary

Transient semantics  
**shallow** types
Asserts in typed code
Gradual Types + Costs

avg : Gradebook -> Num
def avg(g):
    return mean(get_column(g, "score"))

avg(quiz_1_grades)
avg(recipe_book)
avg(42)

Type soundness needs Runtime checks

Costs depend ...

**deep**
- check full gradebook
  - 9x

**shallow**
- check book shape, numbers
  - ~1x
avg : Gradebook -> Num

def avg(g):
    return mean(get_column(g, "score"))

avg(quiz_1_grades)
avg(recipe_book)
avg(42)
avg : Gradebook -> Num
def avg(g):
    return mean(get_column(g, "score"))

avg(quiz_1_grades)
avg : Gradebook → Num
def avg(g):
    return mean(get_column(g, "score"))

depth:
    no boundaries!
    1x

shallow:
    more types, more checks
    2x

avg(quiz_1_grades) ✓
2 modules ➤ deep or shallow (pldi'22)
2 modules ➔ deep or shallow

9 points

(pldi'22)
2 modules ▶ deep or shallow

9 points

Q. where to?

(pldi'22)
3 modules ➤ deep or shallow

27 points
4 modules ➤ deep or shallow

81 points
4 modules $\Rightarrow$ deep or shallow

Q. where to?

Can profilers help?
Profiler

Statistical Profiler

Total cpu time observed: 1192ms (out of 1236ms)
Number of samples taken: 23 (once every 52ms)

<table>
<thead>
<tr>
<th>Idx</th>
<th>Total ms(pct)</th>
<th>Self ms(pct)</th>
<th>Caller Name+src</th>
<th>Callee</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>818(68.6%)</td>
<td>0(0.0%)</td>
<td>??? [12]</td>
<td>evolve [17]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>evolve main</td>
<td>evolve [17]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>shuffle-vector [19]</td>
<td>death-birth [18]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>??? [20]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>match-up* [22]</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>152(12.7%)</td>
<td>152(12.7%)</td>
<td>contract-wrapper</td>
<td></td>
</tr>
</tbody>
</table>

Contract Profiler
Profilers

Statistical Profiler
Total %
Self %

Contract Profiler
Statistical Profiler

Total %  Self %

Contract Profiler

cpu time: 984 real time: 984 gc time: 155
Running time is 18.17% contracts
253/1390 ms

(interface:death-birth pop main)
142 ms

(->* ((cons/c (vectorof automaton?))
    (vectorof automaton?))
  any/c)
  (#:random any/c)
  (cons/c (vectorof automaton?)
    (vectorof automaton?))

(interface:match-up* pop main)
81.5 ms

(-> ....)

(interface:population-payoffs pop main)
29 ms

(-> ....)
Deep types
Contract @ boundary

- Contract %
- Total %
- Self %

Shallow types
Asserts in typed code

- Total %
- Self %
The Problem

Q. where to?

Q. how to find a boundary?

Contract %  Total %  Self %
The Problem

Q. where to?

Q. how to find a boundary?

A. Rational Programmer experiment

Contract %  Total %  Self %
Rational Programmer
Rational Programmer

Identify strategies, let them compete.
Rational Programmer

Identify strategies, let them compete.

Deep ( ▼ ▼ ▼ )

□ □ □ / □ □ □ / □ □ □ ➔ □ □ □
Rational Programmer
Identify strategies, let them compete.

Deep (□■□)      Shallow
□□ / □□ / □□ → □□  ...  → □□
Rational Programmer

Identify strategies, let them compete.

Deep ( yellow blue orange )  Shallow

Type-Aware Deep
1. yellow
2. blue / orange

...
Rational Programmer

Identify strategies, let them compete.

Deep

\[
\begin{align*}
\text{Deep (}\square\square\square\text{)} & \quad \text{Shallow} \\
\square\square/\square\square/\square\square & \quad \ldots
\end{align*}
\]

Type-Aware Deep

1. \square
2. \square/\square

Type-Aware Shallow

\[
\begin{align*}
\ldots & \quad \ldots
\end{align*}
\]
Rational Programmer

Identify strategies, let them compete.

Deep (□□□□□)  
□□ / □□ / □□ → □□  
Shallow

□□  
...  
□□

Type-Aware Deep  
1. □□  
2. □□ / □□ → □□  
Type-Aware Shallow

□□  
...  
□□

Lattice[S; D] count #typed, choose Deep or Shallow
Rational Programmer

Identify strategies, let them compete.

Deep (■■■■)  Shallow

Type-Aware Deep  Type-Aware Shallow

Lattice[S; D] count #typed, choose Deep or Shallow
null, pldi22 baselines
Rational Programmer

Identify strategies, let them compete.
Rational Programmer

Identify strategies, let them compete.

For all starting points,
Goal = path to a fast config
Rational Programmer

Identify strategies, let them compete.

For all starting points,
Goal = path to a fast config

strict = never slow down
k loose = k slower steps

99x ➤ 99x ➤ 3x ➤ 1x
3x ➤ 99x ➤ 1x
Dataset

- 16 GTP Benchmarks
- 116 K starting points
- **1.2 M** measurements
- 5 GB output
- 10 months on CloudLab
How often do the strategies succeed?
How often do the strategies succeed?
How often do the strategies succeed?

\[ X = \text{strategies}, \ Y = \% \text{ scenarios} \]
How often do the strategies succeed?

example data
How often do the strategies succeed?

strict success
How often do the strategies succeed?

strict success

Contract > Statistical [total or self]
How often do the strategies succeed?

strict success

Contract > Statistical [total or self]
Total ~= Self
How often do the strategies succeed?

**strict success**

- Contract > Statistical [total or self]
- Total ~= Self
- Deep >> Shallow

![Bar chart](chart.png)
How often do the strategies succeed?

Strict success:

- Contract > Statistical [total or self]
- Total ~= Self
- Deep >> Shallow

Type-aware, lattice-aware make little difference.
How often do the strategies succeed?

- strict success: 1
- loose: 

![Bar graph showing success rates for different strategies.](image-url)
How often do the strategies succeed?

- **Strict success**: 1
- **1 loose**: 2
- **2 loose**
How often do the strategies succeed?

- Strict success
- 1 loose
- 2 loose
- 3 loose
How often do the strategies succeed?
How often do the strategies succeed?

Looseness helps a bit, profilers rarely benefit from a wrong turn.
How often do the strategies succeed?

- strict success
- 1 loose
- 2 loose
- 3 loose
- N loose
- strict 3x

![Bar chart showing the success rates of different strategies.](chart.png)
How often do the strategies succeed?

strict success | 1 loose | 2 loose | 3 loose | N loose | strict 3x

3x success helps Shallow
* **contract** profiling + **deep** types
  = **best** for type migration

* shallow types do not help
Takeaways

* **contract** profiling + **deep** types = **best** for type migration

* shallow types do not help

Q. hybrid strategies, shallow profilers?
Takeaways

* the **rational programmer** method enables rigorous **experiments**

* **contract** profiling + **deep** types = **best** for type migration

* shallow types do not help

Q. hybrid strategies, shallow profilers?
Takeaways

* the **rational programmer** method enables rigorous **experiments**

* **contract** profiling + **deep** types = **best** for type migration

* shallow types do not help

Q. hybrid strategies, shallow profilers?
Deep -> optimistic
Type-aware D. -> cost-aware optimistic
Shallow -> conservative
Type-aware S. -> cost-aware conservative
Lattice[S,D] -> config-aware
Rand -> null
Pldi22 -> toggle
Skylines per Benchmark
Table 3. How many scenarios can possibly reach 1x without removing types?

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># Scenario</th>
<th>% Hopeful</th>
</tr>
</thead>
<tbody>
<tr>
<td>morsecode</td>
<td>67</td>
<td>100.00%</td>
</tr>
<tr>
<td>forth</td>
<td>76</td>
<td>36.84%</td>
</tr>
<tr>
<td>fsm</td>
<td>62</td>
<td>100.00%</td>
</tr>
<tr>
<td>fsmoo</td>
<td>68</td>
<td>100.00%</td>
</tr>
<tr>
<td>mbta</td>
<td>72</td>
<td>0.00%</td>
</tr>
<tr>
<td>zombie</td>
<td>74</td>
<td>35.14%</td>
</tr>
<tr>
<td>dungeon</td>
<td>242</td>
<td>0.00%</td>
</tr>
<tr>
<td>jpeg</td>
<td>230</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># Scenario</th>
<th>% Hopeful</th>
</tr>
</thead>
<tbody>
<tr>
<td>lnm</td>
<td>295</td>
<td>100.00%</td>
</tr>
<tr>
<td>suffixtree</td>
<td>718</td>
<td>100.00%</td>
</tr>
<tr>
<td>kcfa</td>
<td>2,031</td>
<td>100.00%</td>
</tr>
<tr>
<td>snake</td>
<td>6,559</td>
<td>100.00%</td>
</tr>
<tr>
<td>take5</td>
<td>6,558</td>
<td>0.00%</td>
</tr>
<tr>
<td>acquire</td>
<td>19,532</td>
<td>5.45%</td>
</tr>
<tr>
<td>tetris</td>
<td>18,791</td>
<td>100.00%</td>
</tr>
<tr>
<td>synth</td>
<td>59,046</td>
<td>100.00%</td>
</tr>
</tbody>
</table>
Opt Boundary vs. the others
Type-Aware Boundary vs. the others
Table 4. Which levels of the migration lattice have any acceptable configurations?

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#acceptable</th>
<th>Benchmark</th>
<th>#acceptable by lattice level</th>
</tr>
</thead>
<tbody>
<tr>
<td>morsecode</td>
<td>1 2 4 4 3</td>
<td>lnm</td>
<td>1 9 38 93 138 116 39</td>
</tr>
<tr>
<td>forth</td>
<td>1 2 1 1 0</td>
<td>suffixtree</td>
<td>1 1 0 0 1 4 4</td>
</tr>
<tr>
<td>fsm</td>
<td>1 3 4 7 4</td>
<td>kcfa</td>
<td>1 8 22 33 24 29 15</td>
</tr>
<tr>
<td>fsmoo</td>
<td>1 2 4 2 4</td>
<td>snake</td>
<td>1 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>mbta</td>
<td>1 4 4 0 0</td>
<td>take5</td>
<td>1 2 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>zombie</td>
<td>1 2 3 1 0</td>
<td>acquire</td>
<td>1 8 28 51 45 16 2 0 0 0 0</td>
</tr>
<tr>
<td>dungeon</td>
<td>1 0 0 0 0</td>
<td>tetris</td>
<td>1 12 56 121 169 128 118 133 112 42</td>
</tr>
<tr>
<td>jpeg</td>
<td>1 2 1 1 4 4</td>
<td>synth</td>
<td>1 1 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
</tbody>
</table>
Best-Case Lattice
Best-Case Lattice

fs moo

0.69x
1.03x 7.68x 0.84x 308.73x
7.77x 1.20x 7.46x 319.56x 0.80x 322.58x
7.58x 1.12x 316.50x 0.64x
1.00x
Best-Case Lattice

forth

1.66x

1.03x  1.56x  2.08x  16363.16x

2.30x  2.08x  1.55x  16007.89x  16692.01x  0.97x

1.53x  16185.23x  1.01x  0.95x

1.00x
Best-Case Lattice

mbta

1.23x

1.33x  1.08x  1.24x  1.24x

0.98x  1.24x  1.10x  1.25x  1.11x  1.26x

0.99x  0.97x  1.24x  1.09x

1.00x
## Best-Case Lattice

### dungeon

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.30x</td>
<td>2.33x</td>
<td>1.21x</td>
<td>4.01x</td>
<td>472.06x</td>
<td>1.47x</td>
</tr>
<tr>
<td>2.18x</td>
<td>3.98x</td>
<td>3.97x</td>
<td>492.31x</td>
<td>478.60x</td>
<td>1.56x</td>
</tr>
<tr>
<td>3.91x</td>
<td>483.09x</td>
<td>1.65x</td>
<td>1.58x</td>
<td>2.53x</td>
<td>3.63x</td>
</tr>
<tr>
<td>1.55x</td>
<td>3.62x</td>
<td>497.28x</td>
<td>1.13x</td>
<td>1.12x</td>
<td></td>
</tr>
<tr>
<td>1.00x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Best-Case Lattice

jpeg

0.21x

0.20x 1.37x 23.55x 0.21x 1.65x

1.15x 22.76x 1.38x 0.20x 1.36x 23.27x 1.59x 1.69x 23.19x 1.67x

1.18x 1.20x 23.89x 1.33x 1.67x 23.25x 1.02x 1.61x 1.69x 23.46x

1.19x 0.98x 1.66x 25.81x 1.01x

1.00x
Best-Case Lattice

lnm
0.79x
Best-Case Lattice
Takeaways

* the **rational programmer** method enables rigorous **experiments**

* **contract** profiling + **deep** types = **best** for type migration

* shallow types do not help