How Profilers Can Help Navigate Type Migration

* Ben Greenman
Matthias Felleisen
Christos Dimoulas

oopsa'23
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!

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

Abstract
Gradual systems have come to embrace distributed and typed languages. In many cases, the systems start as innocent prototypes. Soon enough, though, they grow into complex, multi-module programs, at which...
Old Problem, New Idea

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

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

Abstract
In recent years, basic support for typing restriction has been common. For many years, the systems start at innocent preprocessor. Soon enough, though, they grow into complex, multi-module programs, at which...
Old Problem, New Idea

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

**Is Sound Gradual Typing Dead?**
Asana Takakawa, Daniel Felsley, Ben Greenman, Max S. New, Jan Visek, Matthias Felleisen
Northeastern University, Boston, MA

Abstract

Researchers have come to embrace dynamic polymorphism.

In some cases, the systems start as innocent prototypes. Soon enough, though, they grow into complex, multi-module programs, at which point... How to find??

**Rational Programmer method (icfp'21)**
Gradual Types + Costs
def avg(g):
    return mean(get_column(g, "score"))

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

def mean(nums):
    ....

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"))

def mean(nums):
    ....

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

Examples:

```python
avg(quiz_1_grades)
avg(recipe_book)
avg(42)
```

Add types, code still runs.
Gradual Types + Costs

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

```
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  Runtime checks
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 ← 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  ⇐  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 : Gradebook -> Num
def avg(g):
    return mean(get_column(g, "score"))

eval(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 (pldi'22)

9 points
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 ➤ deep or shallow

Q. where to?

Can profilers help?
### Statistical Profiler

<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>evolve [17]</td>
<td>evolve [17]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>evolve main</td>
<td>shuffle-vector [19]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>death-birth [18]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>??? [20]</td>
</tr>
<tr>
<td>[24]</td>
<td>152(12.7%)</td>
<td>152(12.7%)</td>
<td>match-up* [22]</td>
<td>shuffle-vector [19]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>contract-wrapper</td>
<td></td>
</tr>
</tbody>
</table>

**Total cpu time observed:** 1192ms (out of 1236ms)

**Number of samples taken:** 23 (once every 52ms)
Profilers

Statistical Profiler
Total %  Self %

Contract Profiler
Profilers

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
  (-> ....)
<table>
<thead>
<tr>
<th>Deep types</th>
<th>Shallow types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contract @ boundary</td>
<td>Asserts in typed code</td>
</tr>
<tr>
<td>✓ Contract %</td>
<td>✓ Contract %</td>
</tr>
<tr>
<td>✓ Total %</td>
<td>✓ Total %</td>
</tr>
<tr>
<td>✓ Self %</td>
<td>✓ Self %</td>
</tr>
</tbody>
</table>
Q. where to?

Q. how to find a boundary?

Contract %  Total %  Self %
Q. where to?

A. Rational Programmer experiment

Q. how to find a boundary?

Contract %  Total %  Self %
Rational Programmer
Rational Programmer

Identify strategies, let them compete.
Rational Programmer

Identify strategies, let them compete.

Deep ( ▶ ◀ ▶ )

[Diagram showing arrows between different states]
Rational Programmer

Identify strategies, let them compete.

Deep (⟥⟥⟥) / Shallow

...
Rational Programmer

Identify strategies, let them compete.

Deep (■■■■) → Shallow

Type-Aware Deep

1. ■■ / ■■ / ■■ → ■■
2. ■■ / ■■ → ■■
Rational Programmer

Identify strategies, let them compete.

Deep ( ■■■ )

Type-Aware Deep
1. ■
2. ■ / ■

Shallow

... ➔ ■■

Type-Aware Shallow

... ➔ ■■
Rational Programmer

Identify strategies, let them compete.

Deep ( )
Shallow...

Type-Aware Deep 1.
Type-Aware Shallow 2.

Lattice[S; D] count #typed, choose Deep or Shallow
Rational Programmer
Identify strategies, let them compete.

Deep ( )

Shallow

Type-Aware Deep
1. Type-Aware Shallow
2. ...

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 = \textbf{path} to a fast config
For all starting points, Goal = \textit{path} to a fast config

\textbf{strict} = faster each step
\textbf{k loose} = k slower steps
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?
How often do the strategies succeed?

$X =$ strategies, $Y =$ % 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
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
How often do the strategies succeed?

strict success  1 loose  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
How often do the strategies succeed?
How often do the strategies succeed?

3x success helps Shallow
Takeaways
**Takeaways**

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

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

* shallow types do not help
* **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*
Takeaways

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

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

* shallow types do not help
https://github.com/bennn/rational-deep-shallow
Translation: talk -> paper

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
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 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 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 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 1</td>
</tr>
</tbody>
</table>
Takeaways

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

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

* shallow types do not help