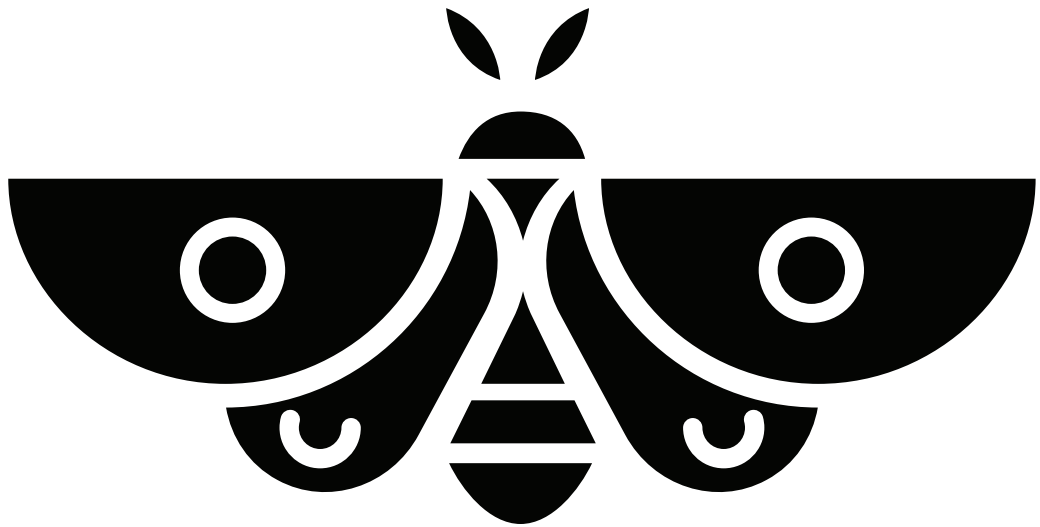
Extending Moth to Support Types as Objects

Erin Greenwood-Thessman, Michael Homer, James Noble

Victoria University of Wellington



What is Moth?

Moth is an interpreter of Grace, an object-oriented language.

It extends SOMns, a Newspeak interpreter using Truffle and GraalVM.

Grace has a gradual type system.

Transient Gradual Typing

For Moth, we implemented transient gradual typing.

```
1   type Point = interface {
2         x
3         y
4   }
5   def position: Point = object {
6         def x: Number = 5
7         def y: Number = 2
8   }
9
10  method equal(a: Point, b: Point) → Boolean {
11        return (a.x == b.x) && { a.y == b.y }
12  }
```

How where types previously supported?

Types are created during translation of the AST and are determined statically with its own name lookup.

Type checks occur as part of writes, reads, and method calls.

To optimize the checks, Moth uses:

- Subtype Cache
- Specialization

Why did we want to change it?

- Unify name lookup for types with rest of Grace
- Support user-defined types

Types as Objects

Made types into objects that can be passed around.

These support method calls like any other object.

Seperated type checks from writes and other operations to their own node in the AST. These nodes self-optimize type checking during execution of the program.

Type as Objects

```
1  type Point = interface { x, y }
2  // Same as
3  method Point { return interface { x, y } }
4
5  def position: Point = Point.cast(object {
6      def x: Number = Number.cast(5)
7      def y: Number = Number.cast(2)
8  })
9
10 method equal(a: Point, b: Point) → Boolean {
11     Point.cast(a)
12     Point.cast(b)
13     return Boolean.cast((a.x == b.x) && { a.y == b.y })
14 }
```

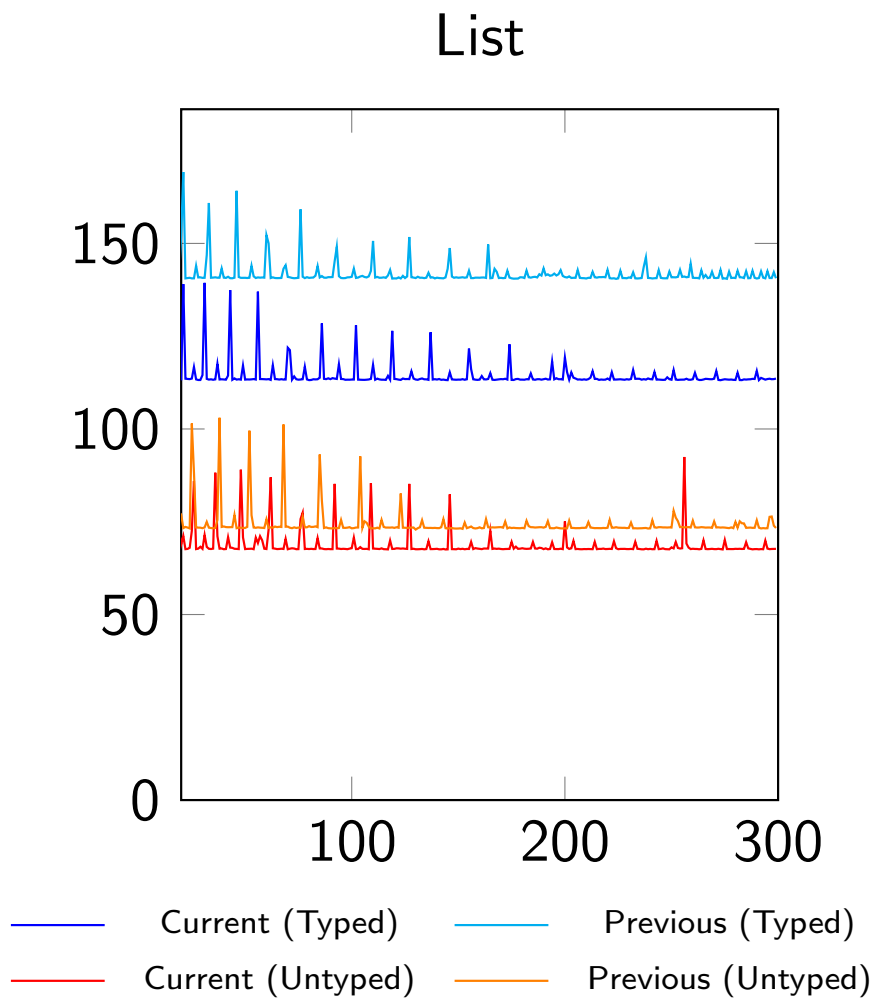
Performance

We ran 21 benchmarks using the new and previous implementations, comparing typed and untyped performance.

The preliminary results show that for the benchmarks:

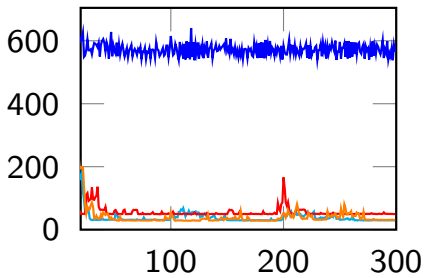
- 7 were slower
- 5 had similar performance
- 8 were faster
- 1 was faster untyped and slower typed

Example Graph

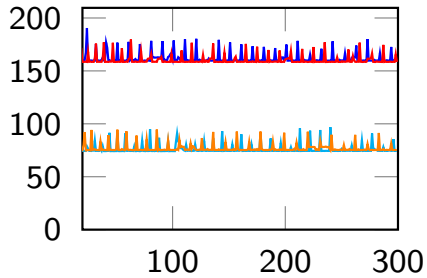


Worst Performance

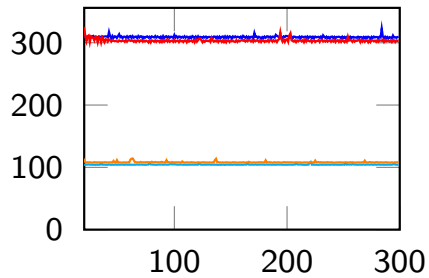
DeltaBlue



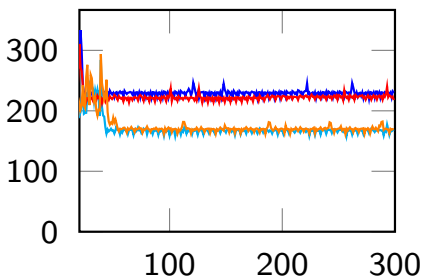
Float



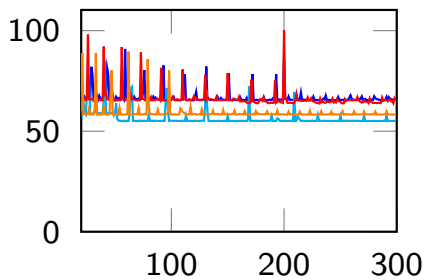
Go



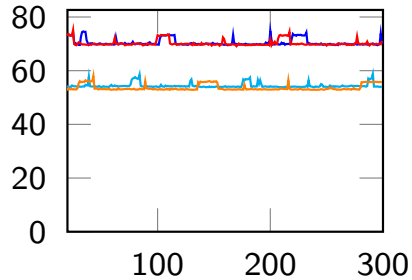
Havlak



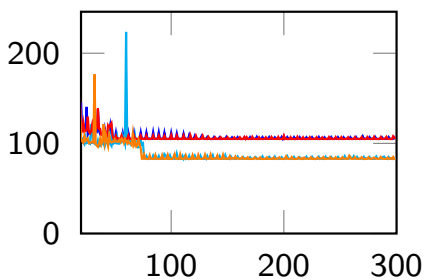
Permute

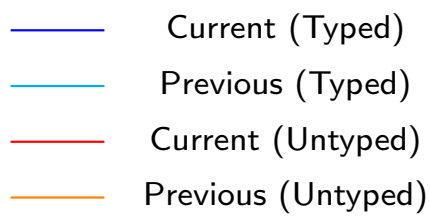


Sieve

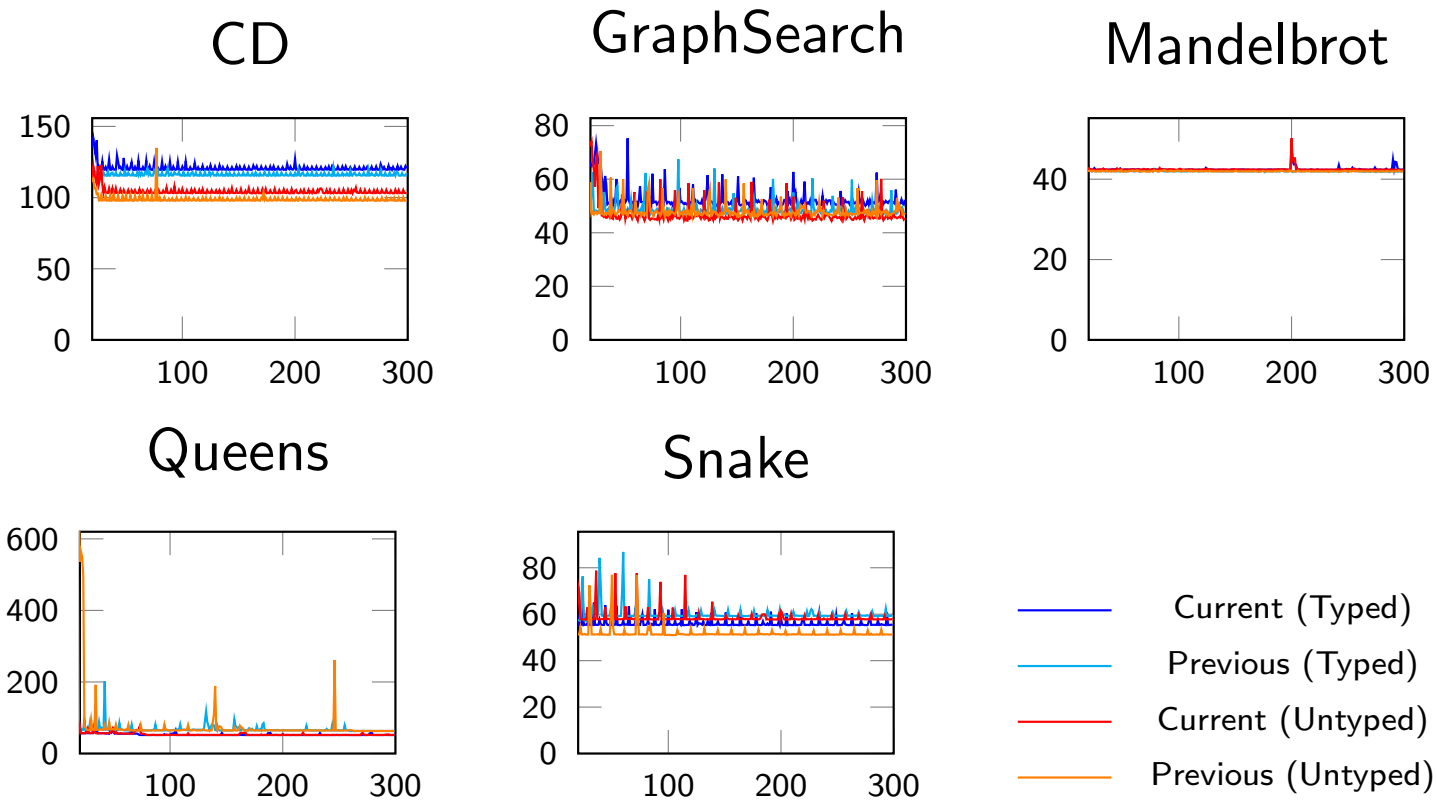


SpectralNorm

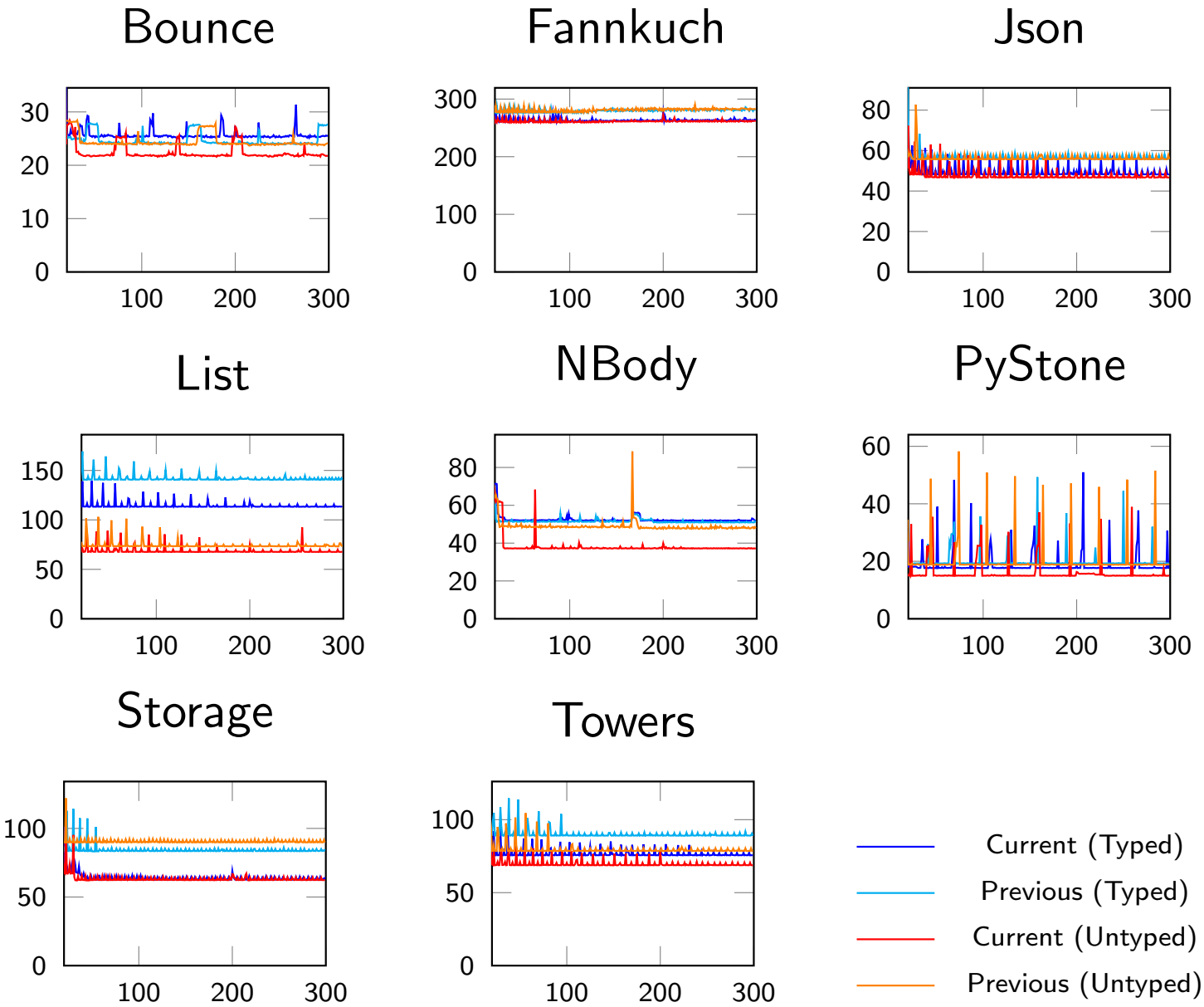




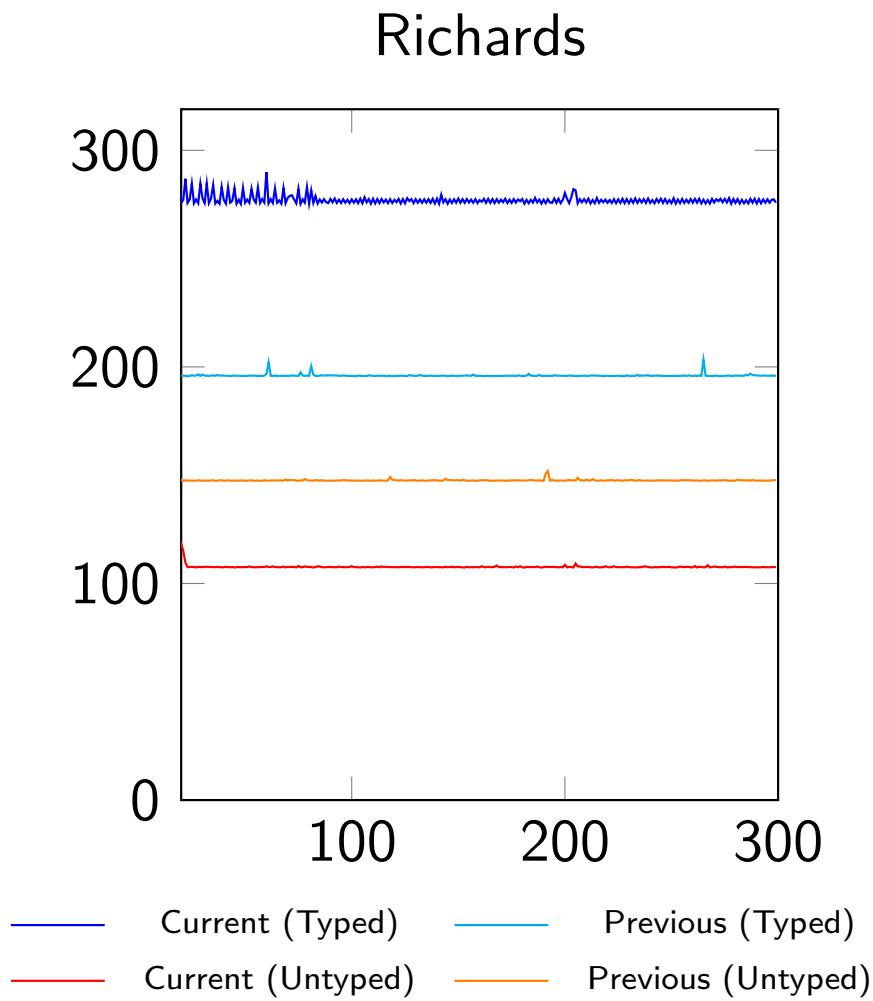
Similar Performance



Faster Performance



The Richards Benchmark



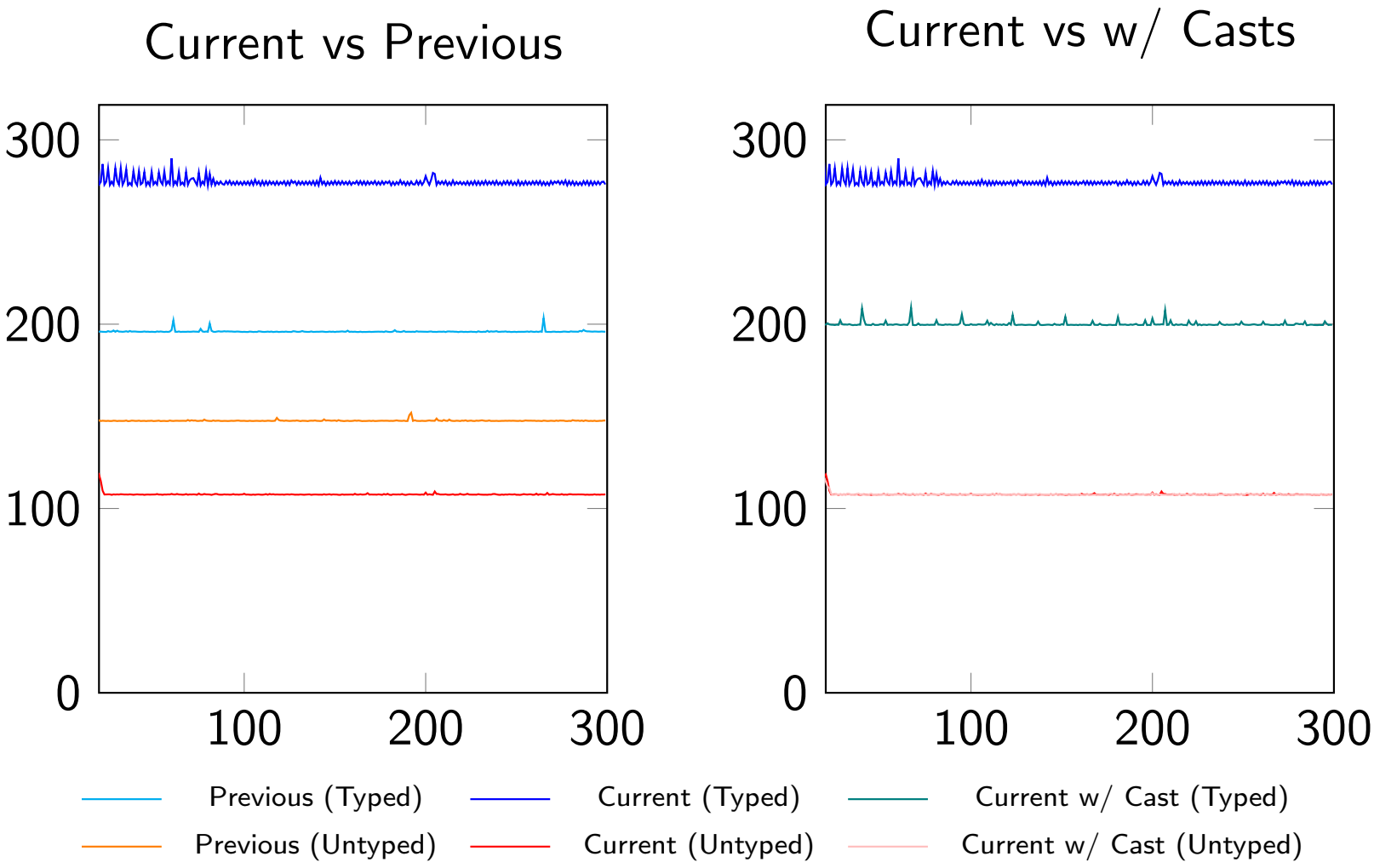
Evaluating Worst-case Performance

We evaluated the worst-case performance of transitive typing by manually inserting type casts everywhere.

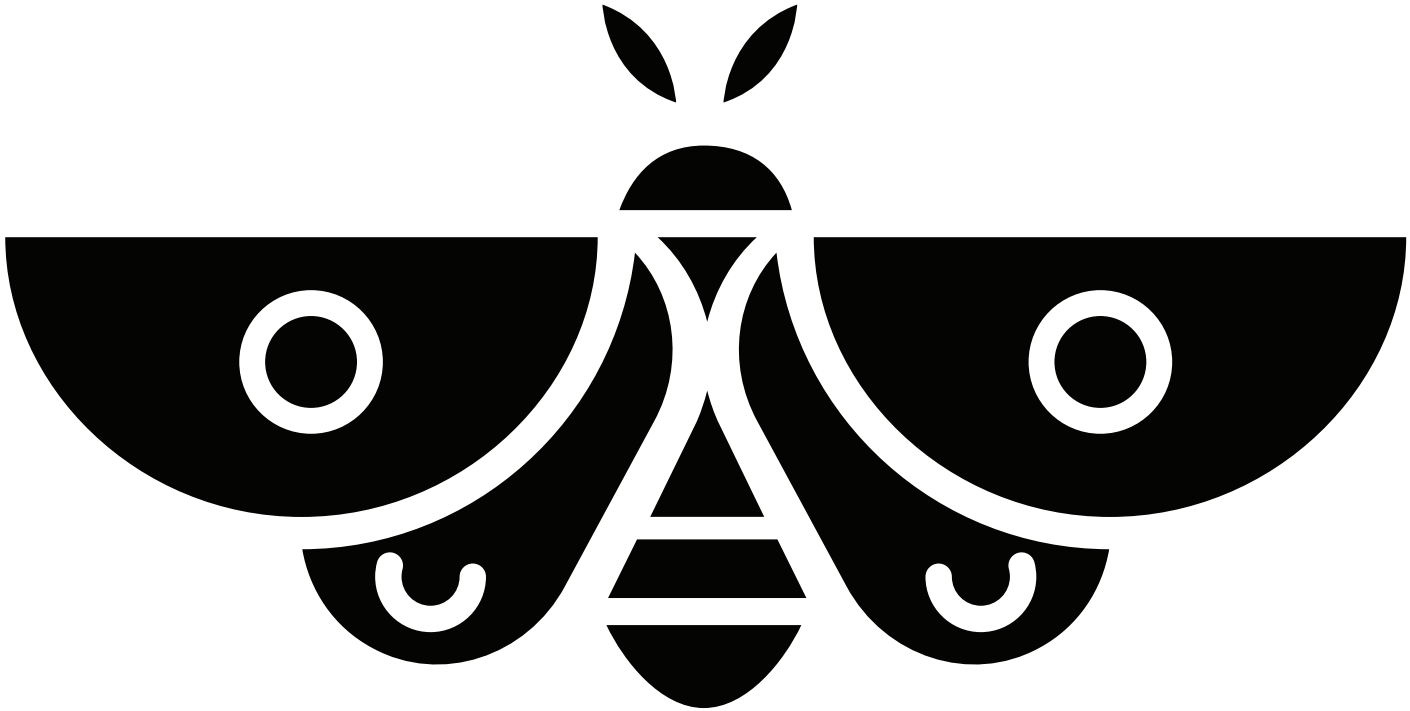
We found that even with extra type checks, the optimized execution had the same performance as with the current type checks on average.

However there was something odd with Richards...

The Richards Benchmark



Questions?



Logo by Jasmine Vollherbst (jasmine@jasminevollherbst.com)