WebAssembly GC Optimization Update

Alon Zakai

Aug 23, 2021

Big Picture

Good benchmark results on Dart.

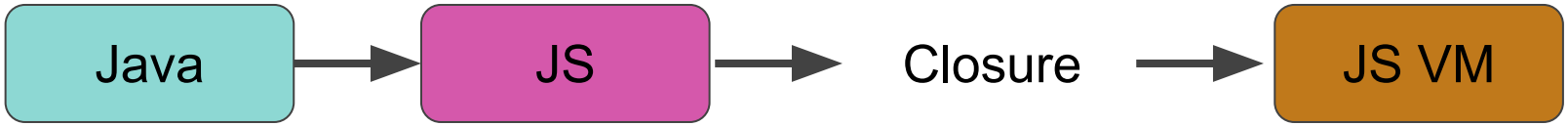
Java (J2CL) results **not** good enough yet.

Wasm GC being fast is a blocker for shipping - or else, why wouldn’t people use JavaScript?

Talk Focus

Mostly on the toolchain.

Need to do the things that both Closure Compiler and the JS VM were doing - much can only be done in the toolchain.



Main Theme

Not really “nominal vs structural.” The key issue is: “**more static type info vs less**.”

Wasm already has lots of useful static type info (e.g. type info for function parameters and results) and is getting even more (e.g. per-table types).

Devirtualization

All calls in Java are **indirect** (virtual). Indirect calls are **not fast**.

Try to turn them into **direct** calls, which can then also be **inlined** & further optimized.

```
(type $A (struct (ref $A.vtable) i32))
(type $B (struct (ref $B.vtable) i32))
(type $C (struct (ref $C.vtable) i32 f64)
         (extends $A))

(type $A.vtable (struct (ref $functype.A)))
(type $B.vtable (struct (ref $functype.B)))
(type $C.vtable (struct (ref $functype.C)
                        (ref ..))
                (extends $A.vtable)))
```

- Differentiating **$A** and **$B** (& their vtables) is useful.
- Knowing that **$C** extends only **$A** is useful.

Devirtualization: Impl

- In the entire module, find all the values written to struct fields, like
  `(struct.new $A.vtable (ref.func $foo))`
  or in `struct.set`, hoping for **constants**.
- Find `struct.get`s and figure out which `struct.new` and `struct.set`s are **relevant** to it.
  - If only a single constant value, **apply it**!

Static Type Info Helps!

```
(type $A.vtable (struct (ref $functype.A)))
(type $B.vtable (struct (ref $functype.B)))
(type $C.vtable (struct (ref $functype.C))
                (extends $A.vtable)))

;; affects $A.vtable, $C.vtable, not $B.vtable
(struct.set $A.vtable ..)

;; affects $B.vtable, not $A.vtable, $C.vtable
(struct.set $B.vtable ..)
```

Devirtualization: Results

Using `--nominal` in Binaryen:

- **41%** speedup.
- **4%** code size reduction.

Cast Elimination?

Some casts needed for validation. Others, not (imagine this is after inlining or such):

```
(func $foo (param $x (ref $A))
           (result   (ref $A))
  (ref.cast
    (local.get $x)
    (global.get $A.rtt)))
```

Cast Elimination?

Even if we write rtt.canon there, we can't remove the cast.

```
(func $foo (param $x (ref $A))
           (result   (ref $A))
  (ref.cast
    (local.get $x)
    (rtt.canon $A))) ;; changed
```

Cast Elimination?

Binaryen introduced a new **TrapsNeverHappen** mode. If set, the optimizer assumes the program will not actually trap.

This is a **13%** speedup and **6%** code size reduction.

```
(func $foo (param $x (ref $A))
           (result   (ref $A))
  (local.get $x)) ;; no cast!
```

Cast Elimination?

Even TrapsNeverHappen does not help with `ref.test`. But **static type information** would!

```
(func $foo (param $x (ref $A))
           (result   (ref $A))
  (ref.test $A ;; => (i32.const 1)
    (local.get $x)))
```

Toolchain-Only?

Wasm object files today have more information than normal object files, like **relocations**.

We could in principle do the same in wasm GC:

Reasons Against Toolchain-Only

Relocations don’t make sense for shipping wasm. But static GC type info does: **Inlining in the VM** for GC, for example.

Simpler overall ecosystem.

Conclusion

More static type information is good!

This is **not** in opposition to structural typing, so long as structural behavior is **in addition**.

Questions?