A New Hope

Compiling Managed Languages to WebAssembly

8 Feb 2024—EPFL

Andy Wingo

Igalia, S.L.

Agenda

WebAssembly, then and now

Experience in Hoot Scheme-to-Wasm compiler

Challenges

Discussion

WebAssembly, the story

WebAssembly is an exciting new universal compute platform

WebAssembly, the pitch

Predictable portable performance

- Low-level
- Within 10% of native

Reliable composition via isolation

- Modules share nothing by default
- No nasal demons
- Memory sandboxing

Compile your code to WebAssembly for easier distribution and composition

WebAssembly, the hype

It’s in all browsers! Serve your code to anyone in the world!

It’s on the edge! Run code close to your users!

It’s the new lightweight virtualization: Wasm is what containers were to VMs! Give me that Kubernetes cash!!!

WebAssembly, the reality

WebAssembly is a weird backend for a C compiler

What about Scala, OCaml, Scheme, and so on – what about *us*?

WebAssembly, the reality (2)

WebAssembly 1.0 was not well-suited to garbage-collected languages

GC and WebAssembly 1.0

Where do garbage-collected values live?

For WebAssembly 1.0, only possible answer: linear memory

```
(module
  (global $hp (mut i32) (i32.const 0))
  (memory $mem 10)) ;; 640 kB
```

Bundle e.g. a bump-pointer allocator

Out of memory? Run GC (which you also bundle)

GC and WebAssembly 1.0 (2)

Problem: Stop-the-world, not parallel, not concurrent, oblivious to system memory pressure

Problem: Maintaining root set introduces overhead

- Live object identification starts with roots: globals and locals from active stack frames
- *WebAssembly gives you no way to visit active stack frames*

Gut check: gut says no

GC and WebAssembly 1.0 (3)

There is already a high-performance concurrent parallel compacting GC in the browser

Halftime: C++ 1 – Scala 0

Hark: GC is here!

WebAssembly now has extension for GC-managed data types

Shipping in Firefox, Chrome; Safari soon

Google Sheets calculation engine: Java to Wasm/GC

Hoot Scheme-to-Wasm compiler: `https://gitlab.com/spritely/guile-hoot`

Small binaries: tens of kilobytes

Interlude

`https://davexunit.itch.io/strigoform`

Bullet-hell vertical scroller written using Hoot

Compiling to Wasm/GC

- **Value representation**
- Retargetting an existing compiler
- Pain points

Scheme Values

```
;;         any  extern  func
;;          |
;;          eq
;;       /  |   \
;; i31 struct  array
```

A reasonable unitype: `(ref eq)`

Immediate values in `(ref i31)`

- fixnums with 30-bit range
- chars, bools, etc

Explicit nullability: `(ref null eq)` vs `(ref eq)`

Scheme Values (2)

```
(rec
 (struct $heap-object
   (sub
    (struct (field $hash (mut i32)))))
 (struct $pair
   (sub $heap-object
     (struct
      (field $hash (mut i32))
      (field $car (mut (ref eq)))
      (field $cdr (mut (ref eq))))))
 ...)
```

Isorecursive subtyping on structs, functions, arrays

Wasm/GC structs closer to shapes than source-language types

Scheme Values (3)

```
(func $cons (param (ref eq)
                   (ref eq))
            (result (ref $pair))
  (struct.new_canon $pair
    ;; Lazily init hash if needed.
    (i32.const 0)
    ;; Car and cdr.
    (local.get 0)
    (local.get 1)))

(func $%car (param (ref $pair))
            (result (ref eq))
  (struct.get $pair 1 (local.get 0)))
```

Scheme Values (4)

```
(func $car (param $arg (ref eq))
           (result (ref eq))
  (block $not-pair
    (return_call
     $%car
     (br_on_cast_fail $not-pair
         (ref eq) (ref $pair)
      (local.get $arg))))
  (call $type-error)
  (unreachable))
```

Compiling to Wasm/GC

- *Value representation*
- **Retargetting an existing compiler**
- Pain points

On retargetting

Ideal: whole-program compiler. Minimize dependencies, dynamic linking

Hoot needed front-end work to enable whole-program compilation

SSA-like IR just fine; “Beyond relooper” great

On retargetting (2)

Backend: Wasm assembly that may reference definitions from a stdlib

Link step to compose stdlib, apply low-level optimizations

Google J2wasm: Emit very naïve Wasm using the text format; rely on Binaryen toolchain to assemble, link, and optimize

Hoot does everything in-house

Compiling to Wasm/GC

- *Value representation*
- *Retargetting an existing compiler*
- **Pain points**

Annoyances

Relative to native, Wasm/GC still missing some pieces

- Varargs
- Async / effect handlers / delimited continuations
- Polymorphism
- Strings

Solution to all these is to virtualize

Annoyances: Varargs

Varargs: uniform calling convention

```
(type $kvarargs
  (func (param $nargs i32)
        (param $arg0 (ref eq))
        (param $arg1 (ref eq))
        (param $arg2 (ref eq))))
```

Additional args spill to global array (table)

Annoyances: Async

Async: CPS-convert the whole thing

All calls tail calls; explicit stack for non-tail calls

Suspend and resume by slicing stack

Stack switching proposal adds 1-shot delimited continuations, removing need for CPS conversion

Annoyances: Polymorphism

Scheme: Numeric tower (exact integers, fractions, inexact reals, inexact complex)

Wasm: Fast path for fixnums, stdlib for everything else; compiler can unbox sometimes

Would ideally want inline caches that feed into optimizing compiler, but no JIT

Relative to native, Wasm/GC still missing some pieces

Annoyances: Strings

All languages have strings

Sure would be nice to just pass them by reference, use host string support libraries (e.g. regexes)

Instead we ship duplicate, worse versions than what the browser has

Compiling to Wasm/GC

- *Value representation*
- *Retargetting an existing compiler*
- *Pain points*

A New Hope

WebAssembly is now a good target for managed languages

Hoot shows good results are possible

Next up, Scala? Discuss!

```
(visit-links
 "gitlab.com/spritely/guile-hoot"
 "wingolog.org"
 "wingo@igalia.com"
 "igalia.com"
 "mastodon.social/@wingo")
```