A world to win

WebAssembly for the rest of us

17 Mar 2023 – BOB 2023

Andy Wingo

Igalia, S.L.

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 from your web site close to your users!

Compose a library (eg: Expat) into your program (eg: Firefox), without risk!

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

Only some source languages are having success on WebAssembly

What about Haskell, Ocaml, Scheme, F#, and so on – what about *us*?

Are we just lazy? (Well…)

WebAssembly, the reality (2)

WebAssembly (1.0, 2.0) is not well-suited to garbage-collected languages

Let’s look into why

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
```

```
(func $alloc (param $size i32) (result i32)
  (local $ret i32)
  (loop $retry
    (local.set $ret (global.get $hp))
    (global.set $hp
      (i32.add (local.get $size) (local.get $ret)))

    (br_if 1
      (i32.lt_u (i32.shr_u (global.get $hp) 16)
                (memory.size))
      (local.get $ret))

    (call $gc)
    (br $retry)))
```

GC and WebAssembly 1.0 (2)

What hides behind `(call $gc)`?

Ship a GC over linear memory

Stop-the-world, not parallel, not concurrent

But… **roots**.

GC and WebAssembly 1.0 (3)

Live objects are

- the roots
- any object referenced by a live object

Roots are globals and locals in active stack frames

No way to visit active stack frames

GC and WebAssembly 1.0 (3)

Workarounds

- handle stack for precise roots
- spill all possibly-pointer values to linear memory and collect conservatively

Handle book-keeping a drag for compiled code

GC and WebAssembly 1.0 (4)

Cycles with external objects (e.g. JavaScript) uncollectable

A pointer to a GC-managed object is an offset to linear memory, need capability over linear memory to read/write object from outside world

No way to give back memory to the OS

Gut check: gut says no

GC and WebAssembly 1.0 (5)

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

Halftime: C++ 1 – Altlangs 0

Change is coming!

Support for built-in GC set to ship in Q4 2023

With GC, the material conditions are now in place

Let’s compile *our* languages to WebAssembly

Scheme to Wasm

Spritely + Igalia working on Scheme to WebAssembly

Avoid truncating language to platform; bring whole self

- **Value representation**
- Varargs
- Tail calls
- Delimited continuations
- Numeric tower

Scheme to Wasm: Values

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

The 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 to Wasm: Values (2)

Heap objects subtypes of `struct`; concretely:

```
(struct $heap-object
  (struct (field $tag-and-hash i32)))
(struct $pair
  (sub $heap-object
    (struct i32 (ref eq) (ref eq))))
```

GC proposal allows subtyping on structs, functions, arrays

Structural type equivalance: explicit tag useful

Scheme to Wasm: Values (3)

```
(func $cons (param (ref eq)
                   (ref eq))
            (result (ref $pair))
  (struct.new_canon $pair
    ;; Assume heap tag for pairs is 1.
    (i32.const 1)
    ;; Car and cdr.
    (local.get 0)
    (local.get 1)))

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

```
(func $car (param (ref eq)) (result (ref eq))
  (local (ref $pair))
  (block $not-pair
    (br_if $not-pair
      (i32.eqz (ref.test $pair (local.get 0))))
    (local.set 1 (ref.cast $pair) (local.get 0))
    (br_if $not-pair
      (i32.ne
        (i32.const 1)
        (i32.and
          (i32.const 0xff)
          (struct.get $heap-object 0 (local.get 1)))))
    (return_call $%car (local.get 1)))

  (call $type-error)
  (unreachable))
```

Scheme to Wasm

- *Value representation*
- **Varargs**
- Tail calls
- Delimited continuations
- Numeric tower

Scheme to Wasm: Varargs

```
(list 'hey)       ;; => (hey)
(list 'hey 'bob) ;; => (hey bob)
```

Problem: Wasm functions strongly typed

```
(func $list (param ???) (result (ref eq))
  ???)
```

Solution: Virtualize calling convention

```
;; "Registers" for args 0 to 3
(global $arg0 (mut (ref eq)) (i31.new (i32.const 0)))
(global $arg1 (mut (ref eq)) (i31.new (i32.const 0)))
(global $arg2 (mut (ref eq)) (i31.new (i32.const 0)))
(global $arg3 (mut (ref eq)) (i31.new (i32.const 0)))

;; "Memory" for the rest
(type $argv (array (ref eq)))
(global $argN (ref $argv)
        (array.new_canon_default
          $argv (i31.const 42) (i31.new (i32.const 0))))
```

Uniform function type: argument count as sole parameter

Callee moves args to locals, possibly clearing roots

Scheme to Wasm

- *Value representation*
- *Varargs*
- **Tail calls**
- Delimited continuations
- Numeric tower

Scheme to Wasm: Tail calls

```
;; Call known function
(return_call $f arg ...)

;; Call function by value
(return_call_ref $type callee arg ...)
```

Scheme to Wasm

- *Value representation*
- *Varargs*
- *Tail calls*
- **Delimited continuations**
- Numeric tower

Scheme to Wasm: Prompts (1)

Problem: Lightweight threads/fibers, exceptions

Possible solutions

- Eventually, built-in coroutines
- `https://github.com/WebAssembly/binaryen`'s asyncify (not yet ready for GC); see Julia
- **Delimited continuations**

"Bring your whole self"

Scheme to Wasm: Prompts (2)

Prompts delimit continuations

```
(define k
  (call-with-prompt 'foo
    ; body
    (lambda ()
      (+ 34 (abort-to-prompt 'foo)))
    ; handler
    (lambda (continuation)
      continuation)))

(k 10)         ;; ⇒ 44
(- (k 10) 2) ;; ⇒ 42
```

k is the _ in `(lambda () (+ 34 _))`

Scheme to Wasm: Prompts (3)

Delimited continuations are stack slices

Make stack explicit via minimal continuation-passing-style conversion

- Turn all calls into tail calls
- Allocate return continuations on explicit stack
- Breaks functions into pieces at non-tail calls

Scheme to Wasm: Prompts (4)

Before a non-tail-call:

- Push live-out vars on stacks (one stack per top type)
- Push continuation as funcref
- Tail-call callee

Return from call via pop and tail call:

```
(return_call_ref (call $pop-return)
                 (i32.const 0))
```

After return, continuation pops state from stacks

Scheme to Wasm: Prompts (5)

`abort-to-prompt`:

- Pop stack slice to reified continuation object
- Tail-call new top of stack: prompt handler

Calling a reified continuation:

- Push stack slice
- Tail-call new top of stack

No need to wait for effect handlers proposal; you can have it all now!

Scheme to Wasm

- *Value representation*
- *Varargs*
- *Tail calls*
- *Delimited continuations*
- **Numeric tower**

Scheme to Wasm: Numbers

Numbers can be immediate: fixnums

Or on the heap: bignums, fractions, flonums, complex

Supertype is still `ref eq`

Consider imports to implement bignums

- On web: `BigInt`
- On edge: Wasm support module (mini-gmp?)

Dynamic dispatch for polymorphic ops, as usual

Scheme to Wasm

- *Value representation*
- *Varargs*
- *Tail calls*
- *Delimited continuations*
- *Numeric tower*

Miscellenea

Debugging: The wild west of DWARF; prompts

Strings: `stringref` host strings spark joy

JS interop: Export accessors; Wasm objects opaque to JS. `externref`.

JIT: A whole 'nother talk! `https://wingolog.org/archives/2022/08/18/just-in-time-code-generation-within-webassembly`

AOT: wasm2c

WebAssembly for the rest of us

With GC, WebAssembly is now ready for us

Getting our languages on WebAssembly now a S.M.O.P.

Let’s score some goals in the second half!

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