Scheme to Wasm

Use and misuse of the GC proposal

18 Apr 2023 – Wasm GC subgroup

Andy Wingo

Igalia, S.L.

Prehistory

Guile co-maintainer (`https://gnu.org/s/guile/`)

Wanted to target wasm for a while; didn't because no gc

Also didn't know how to do delimited continuations

Now: idea and funding via `https://spritely.institute/`

Interruptions welcome

A work in progress

Spritely + Igalia working on Scheme to WebAssembly

Based on Guile

Re-use front and middle-end, replace backend and runtime

Source IR: “CPS soup” `https://www.gnu.org/software/guile/manual/html_node/CPS-Soup.html`

Data types: f64, i64, u64, SCM

Ark rather than raft

Early days

Scheme to Wasm

Avoid truncating language to platform; bring whole self

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

Scheme to Wasm: Values

The unitype: `(ref eq)`

Non-nullable

Immediate values in `(ref i31)`

- fixnums with 30-bit range
- chars, 2 bools, 3 other oddballs

Scheme to Wasm: Values (2)

Heap objects subtypes of `struct`; concretely:

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

Hybrid nominal typing via `rec`

Scheme to Wasm: Values (3)

```
(func $car (param (ref eq))
           (result (ref eq))
  (struct.get $pair 1
    (block (ref $pair)
      (br_on_cast $pair 0 (local.get 0))
      (call $type-error)
      (unreachable))))
```

`set-car!` checks for `$mutable-pair`; similar treatment for vectors, bytevectors, bitvectors, strings (ugh)

Scheme to Wasm: Values (4)

```
(rec
  ...
  (type $bignum
    (sub $heap-object
      (struct
        (field $hash (mut i32))
        (field $val (ref extern)))))
  ...)
```

Scheme to Wasm

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

Scheme to Wasm: Varargs (1)

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

Problem: Wasm functions strongly typed

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

Solution: Virtualize calling convention

```
;; nargs param; first 3 args as params
(type $kvarargs
  (func (param $nargs i32)
        (param $arg0 (ref eq))
        (param $arg1 (ref eq))
        (param $arg2 (ref eq))))

;; next 5 args as globals
(global $arg3 (mut (ref eq)) (i31.new (i32.const 0)))
...
(global $arg7 (mut (ref eq)) (i31.new (i32.const 0)))

;; "Memory" for the rest
(table $argv (ref eq) 0 (i31.new (i32.const 0)))
```

Downside: export/import globals, table; globals worth it?

```
(define (pi pair)
  (values (car pair) (cdr pair)))
(define (dup pair)
  (call-with-values (lambda () (pi pair))
    (lambda (car cdr)
      (cons car cdr))))

;; values ignored in for-effect context; equivalent:
(begin (pi pair) #t)
(call-with-values (lambda () (pi pair))
  (lambda args #t))

;; sloppy truncation
(define (car pair) (values (pi pair)))
```

How? Answer in a minute

Scheme to Wasm

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

Scheme to Wasm: Tail calls

Tears of joy

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

If cont not lexically used: escape-only (exception building block)

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 $kvarargs (i32.const 0)
                           val0 val1 val2
                           (call $pop-return))
```

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

Willing to sacrifice multi-shot to use effect handlers proposal, though!

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: DWARF; prompts

Wasm parser, assembler, etc in Scheme (including all V8 extensions)

Strings: `stringref`

“Beyond relooper”; irreducible CFG TBD

No linear memory

AOT: wasm2c

Status: very early days

Stringref usage

```
(type $string
  (sub $heap-object
    (struct
      (field $hash (mut i32))
      (field $str (mut string)))))
```

WTF-8 view for port (like `FILE*`) buffer

Codepoint iter view for `(string-ref str N)`

`string.const` has been a debugging delight

Scheme to Wasm

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