Tail Calls for Wasm

Proposal update

Andreas Rossberg

Dfinity

Recap

Support proper tail calls

…for languages

…for implementation techniques

Property: unbounded sequence of tail calls do not blow stack

Implementation may “amortise”,
i.e., finitely many tail calls use stack space

Proposal Overview

Introduce tail version of each call instruction

Essentially, return combined with call

return_call <*funcidx*>
return_call_indirect <*tableidx*> <*typeidx*>

Text Format

Exactly analogous to **call**/**call_indirect**

…same immediates

…same sugar

Binary Format

Use free opcodes next to existing calls

0x0F **return**
0x10 **call**
0x11 **call_indirect**
0x12 **return_call**
0x13 **return_call_indirect**
0x14-0x19 (reserved)

Execution

Like combination of return with call

Hence unwind operand stack,
keeping only arguments for call

New call frame replaces existing one

Validation

Like combination of return and call

Hence stack-polymorphic

Return type of callee must match caller's

Open question: differentiate function types?

Function Type Attributes?

1. Mark tail-callers?
2. Mark tail-callees?
3. Both?
4. Neither?

Marking Tail-Callers/Callees

Allows separate calling convention

Marking tail-callers potentially saves a register for other calls

Benefit for marking tail-callees?

Drawbacks

Creates function space schisma
(neither type is a subtype of the other)

Need to anticipate all uses of a function
(generally impossible)

In practice, a compiler will pick one

Problem with interop, for libraries, for APIs

Drawbacks

Problem aggravated with function references

No way to convert one type of reference into the other!

Example

```
;; Library A
(type $proc (func no-tail-call))
(func (export "f") (result (ref $proc)) ...)

;; Library B
(type $proc (func may-tail-call))
(func (export "g") (param (ref $proc)) ...)

;; Module C
(func $f (import "A" "f") ...)
(func $g (import "B" "g") ...)
(func (call $g (??? (call $f))))
```

Drawbacks

Problem aggravated with function references

No way to convert one type of reference into the other!

Would need primitive conversion operator

Allocates a form of closure ⇒ requires GC!

Discuss!

Proposal Status

prose spec : (✓)

formal spec : (✓)

interpreter : (✓)

tests : (✓)

(basic proposal)

Implementation Status

V8 : □

SpiderMonkey : □

JSC : □

Chakra : □

Remaining steps

Review PR

Resolve function type question

Implementations