Typed Function References for Wasm

Proposal update

Andreas Rossberg
Dfinity

WA

Motivation

Efficient indirect calls without runtime checks

First-class function pointers without tables

Easy, safe, efficient host interop

Optionally, safe, opaque closures

Split from GC proposal, independently useful

Summary

Based on reference types proposal

Refine **funcref** to fully typed references, non/nullable

Refine **func.ref** to return typed reference

Add check-free **call_ref** instruction

Optionally, **func.bind** instruction for forming closures

Recent Changes

Summarise details of extension

Incorporated nullable **optref** type and respective instructions

```
(type $i32-i32 (func (param i32) (result i32)))

(func $ho (param $f (ref $i32-i32)) (result i32)
    (call_ref (local.get $f) (i32.const 1))
)

(func $inc (param i32) (result i32)
    (i32.add (local.get 0) (i32.const 1))
)

(func $caller (result i32)
    (call $ho (func.ref $inc))
)
```

function references

ref.func $f : [] → [(ref $t)]
where $f : $t

call_ref : [(ref $t) t_1*] → [t_2*]
where $t = [$t_1$*] → [$t_2$*]

return_call_ref : [(ref $t) t_1*] → [t_2*]
where $t = [$t_1$*] → [$t_2$*]

Optional References

Regular (**ref** $t) type does not allow **null**

Separate (**optref** $t) does include **null**

Instructions for checking and converting

Optional References

ref.is_null :	[anyref] → [i32]	(*)
ref.as_non_null :	[(optref $t)] → [(ref $t)]	
br_on_null $l :	[(optref $t)] → [(ref $t)] iff label $l : []	

(*) from reference types proposal

reference subtyping

ref $t

<:

optref $t

<:

funcref

<:

anyref

function references

ref.func $f : $[] \to [(\text{ref } \$t)]$
where $\$f : \$t$

call_ref : $[(\text{optref } \$t)\ t_1{*}] \to [t_2{*}]$
where $\$t = [t_1{*}] \to [t_2{*}]$

return_call_ref : $[(\text{optref } \$t)\ t_1{*}] \to [t_2{*}]$
where $\$t = [t_1{*}] \to [t_2{*}]$

Closures: Motivation

Can roll your own closures, but not interoperably

- type incompatible with regular function refs

- not opaque, exposes closure environment

Not safe/secure to pass to host or other modules

Functions *are* closures already (over instance)

Closures: Summary

Add **func.bind** instruction for partial application

Yields fresh function reference with fewer args

Interchangeable with other function references

Note: cannot construct cycles, RC is enough

```
(type $i32-i32 (func (param i32) (result i32)))

(func $add (param i32 i32) (result i32)
  (i32.add (local.get 0) (local.get 1))
)

(func $mk-adder (param i32) (result (ref $i32-i32))
  (func.bind $i32-i32 (func.ref $add) (local.get 0))
)


(call_ref (call $mk-adder (i32.const 2)) (i32.const 3))
```

closures

call_ref : $[(\text{optref }\$t)\ t_1^*] \rightarrow [t_2^*]$
where type $\$t = [t_1^*] \rightarrow [t_2^*]$

func.bind $\$t'$: $[(\text{optref }\$t)\ t_1^*] \rightarrow [(\text{ref }\$t')]$
where type $\$t = [t_1^*\ t'_1{}^*] \rightarrow [t_2^*]$
type $\$t' = [t'_1{}^*] \rightarrow [t_2^*]$