Typed (Function) References

Proposal status update

Andreas Rossberg

Reference Types Refactored

$$(\textbf{ref}\ \underbrace{\textit{heaptype}}_{\textbf{func}\ |\ \textbf{extern}\ |\ \$t})$$

Reference Types Refactored

$$(\textbf{ref}\ \overbrace{\textbf{null}^{?}}^{\text{nullability}}\ \underbrace{\mathit{heaptype}}_{\textbf{func}\ \mid\ \textbf{extern}\ \mid\ \$t})$$

“**null** ∪ *heaptype*”

Shorthands

funcref = (**ref null func**)

externref = (**ref null extern**)

Subtyping

(**ref** *ht*) <: (**ref null** *ht*)

$t <: **func**

Null References

ref.null *ht* :	[] → [(ref null *ht*)]
ref.is_null :	[(ref null *ht*)] → [i32]
ref.as_non_null :	[(ref null *ht*)] → [(ref *ht*)]
br_on_null $/ :	[(ref null $*t*)] → [(ref $*t*)]
br_on_non_null $/ :	[(ref null $t)] → []

Function References

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

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

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

(Deferring **func.bind** to separate proposal)

Defaultability

Locals and tables rely on default initialisation

Only nullable references have default value

Can’t have non-defaultable locals or tables

Non-Defaultable Locals

1. Original proposal: block-scoped locals

   let (**local** <*valtype*>*) <*instr*>* **end**

   Pros: compositional; no uninitialised locals

   Cons: index shifts; spurious block structure

2. Variation: block-initialised locals

   let (**local** <*localidx*>*) <*instr*>* **end**

   Pros: no index shifts

   Cons: spurious block structure; must track/check initialisation status

Non-Defaultable Locals

3. Minimal: tracked locals

   local.set marks variable as initialised until end of block

   Pros: no index shifts; no spurious blocks

   Cons: must track initialisation status; more costly validation (+4-7%)

   Risk: slippery slope towards more flow-based analysis
   (extending block types would avoid that)

4. Variation: tracked locals with separate initialisation instruction

   local.init marks variable as initialised until end of block

   Pros: as before, but also, initialisation point is easier to identify

   Cons: one extra instruction with identical behaviour

Non-Defaultable Tables

Tables currently require nullable references

Plan: add explicit init value to table definitions

(**table** $t 10 funcref (**ref.null func**))

…may be omitted if type is defaultable

Status

✅ Specification (minus **let**)
…one pending PR

✅ Implementation in reference interpreter

✅ Test suite

✅ Implemented in V8 & SM

Stage 2 (2020/06), mainly let discussion since

Discussion

Adopt tracked locals?

(interpreter implementation and tests already exist on branch)

Anything left after that to move to stage 3?