Reference Types for Wasm

Proposal update

Andreas Rossberg

Dfinity

value types

number types

transparent

storable in memories

i32 i64 f32 f64

reference types

opaque

storable in tables

anyref

funcref exnref ...

Recap: Instructions

ref.null – produce null value

ref.is_null – check for null

ref.func – produce function reference

table.get – load reference from table

table.set – store reference into table

Recap: Bulk Instructions

table.fill – fill table range with ref value

table.size – inquire table size

table.grow – increase table size

Recap: Modified Instructions

table.init – table index immediate

table.copy – table index immediate

call_indirect – table index immediate

select – new version with type immediate

Recent Changes

Require ref.func pre-declare via elem segments

Added nullref as explicit type

Minor syntax tweaks in text format

Recent Spec Work

Mutual dependency with bulk instructions, coherent spec needs both extensions

Consolidated bulk instructions proposal, favouring gaps over hacks

Rebased this repo on bulk repo, filled in all gaps plus cross-product

Coherent combined spec in this repo

Various minor fixes and todos addressed

Proposal Status

prose spec : ✓

formal spec : ✓

Stage 3

interpreter : ✓

tests : ✓ (minus multi-table bulk @lars)

JS API : ✓ (minus some tweaks @wingo)

Implementation Status

V8 : ✓

SpiderMonkey : ✓

JSC : □

Open Suggestion

Remove subtyping from anyref to funcref?

Pro: allows different representations
(e.g., flat fat function pointers)

Con: requires design changes to other
proposals, e.g. bulk instructions, C API

Impact on language

Implies different null values,
need to remove uniform nullref type

Instead, type-annotate **ref.null** instruction

e.g., (**ref.null func**) vs (**ref.null any**)

later also (**ref.null** $T)

Unfortunate redundancy in elem segments

(**elem** T (**ref.null** T) (**ref.null** T) (**ref.null** T))

…unless we fundamentally change validation

What about other ref types, e.g., exnref?

Impact on JS API

Details of boxing functions in ToJsValue?

Impact on C/C++ API

Uses type hierarchy for ref values

```
class Ref
  class Extern
    class Func
    class Memory
    class Table
    class Global
    ...
  class Foreign
  ...
```

```
union Val {
    int32_t i32;
    int64_t i64;
    float32_t f32;
    float64_t f64;
    Ref* ref;
}
```

`Instance::make(Store*, Module*, Extern*[]) -> Instance*`

Motivation: extensible, allows 1st-class handling matching JS

Alternatives

Could defer to post-MVP as additional value type, e.g., flatref

Note: previous resolution at Lyon meeting

Poll

Move to stage 4?