Reference Types for Wasm

Proposal update

Andreas Rossberg

Dfinity

Recap

Add opaque reference type

New form of value type

…usable for locals, globals, parameters, results

…can be put into tables

…can be passed from/to embedder

Can not be constructed or accessed in Wasm

Allow multiple tables

Main motivation: host objects

Recent Changes

Incorporated table.fill, table.grow, table.size

Added type-annotated select instruction and bottom type to avoid lubs and glbs

Finalised opcode assignments

Decision to require forward declaration for uses of **ref.func** (issue #31)

value types

number types

transparent

storable in memories

i32 i64 f32 f64

reference types

opaque

storable in tables

anyref

funcref ...

New 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

Instructions adopted from Bulk Data Proposal

table.fill – fill table range with ref value

table.size – inquire table size

table.grow – increase table size

Modified Instructions

table.init – table index immediate

table.copy – table index immediate

call_indirect – table index immediate

select – new version with type immediate

Subtyping vs joins & splits

data flow joins and splits generally require computing least upper or greatest lower bound

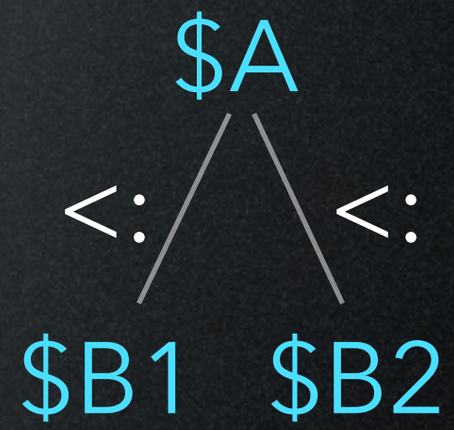
1) **select** – result type is lub of operand types

2) **br_table** – operand type is glb of label types

enough type annotations everywhere else

select

(**local** $x (ref $B1))
(**local** $y (ref $B2))



(**select** (**local.get** $x) (**local.get** $y) (…)) : ?

canonical answer would be (ref $A)

but might be costly to infer in general

Avoiding lubs

solution: make example invalid, require type annotation

new opcode for (**select** <*valtype*>)

works for all value types

(could also be generalised to multi-value)

existing (**select**) only allows numeric types, which have no subtyping

Avoiding lubs

select <*t*> : [<*t*> <*t*> i32] → [<*t*>]

select : [<*t*> <*t*> i32] → [<*t*>]
iff <*t*> <: <*numtype*>

br_table


```
(block (result (ref $A1))
  (block (result (ref $A2))
    (unreachable)
    (br_table 0 1 1 (i32.const 1))
  )
  ...
)
```

type-checks if we assume (ref $B)

but must check that glb exists!

Counter example

```
(block (result f64)
    (block (result f32)
        (unreachable)
        (br_table 0 1 1 (i32.const 1))
    )
    …
)
```

Avoiding glbs

solution: make bottom type a proper citizen

i.e., “officially” allow it in the type system

previously, it was merely an auxiliary device of the checking algorithm

no change to validators except for **br_table**

allows counter example and other useless ones

value types

number types

transparent

storable in memories

reference types

opaque

storable in tables

i32 i64 f32 f64

anyref

funcref

...

(bottom)

Proposal Status

prose spec : ✓*

formal spec : ✓*

interpreter : ✓* Stage 3

tests : ✓

JS API : ✓

* minus declaring ref.func

Implementation Status

V8 : ✓

SpiderMonkey : (✓)

JSC : □

Chakra : □

Remaining steps

Add declared element segment
(blocked on resolving segment format)

…and check for **ref.func**

Finish implementations

JS API Extensions

any JS value can be passed as **anyref**

Wasm exported function or `null` can be passed as **funcref**

`Table#grow` takes optional init value param