Reference Types and Subtyping

Andreas Rossberg

Dfinity

Background

Reference types were originally split off from GC

Turned into longer road map of individual proposals

… (1) reference types

… (2) function references

… (3) type imports

…

Design choices were made to fit that context

Some may no longer appear individually useful

Basic Assumptions

1. Default uniform reference representation
    - established practice in all existing Wasm engines and the vast majority of other GC runtimes
    - simplifies various design aspects
        - avoids arbitrary bias (which types can be im/exported?)
        - doesn’t preclude later addition of alternative representations!
2. Subtyping
    - natural and useful on representation types
    - required for various encodings
    - this proposal introduces the bare minimum, nothing fancy at all

Some dependent features

Function references

…fundamentally require subtyping for nullability, interaction with tables, backwards compatibility of ref.func instruction

Type imports

…express constraints on imports, partial abstraction, avoid type bias

Reference equality

…depends on eqref supertype, essentially a limited anyref

GC types

…subtyping comes up with many use cases,
anyref as escape hatch, e.g. for compiling uniform representations

C/C++ API

…uniform handling of references

Why Not Defer?

Implications on other proposals are poorly understood

Substantial churn

Introduces permanent warts along the way

Last-minute change of basic design choice

Impact on this proposal

Not a small change!

Bare minimum spec change set (link)

…remove anyref subtyping, remove nullref type, change ref.null instruction, introduce type-indexed null values (or lose principal types), change typing of some instructions to work without subtyping, adjust JS API, …

Does not yet include necessary clean-up!

…remove now dysfunct subtyping rules
…revert typing algorithm
…rename anyref type

Impact on other proposals

Bulk instructions (elem segment format)

Exceptions (rules for exnref)

Function references (everything)

Type imports (basic approach)

C/C++ API (needs to change)

Summary

Points against dropping/deferring subtyping:
- substantial churn for many tools and other proposals
- introduces some permanent warts
- leaves C/C++ API in limbo state
- not clear what could change, no alternative road map
- general risk of last-minute design change

Points in favour:
- more time to chew on it, keeps design space open for a little longer