Status Update on the GC Proposal

…and some experiments…

Andreas Rossberg

Dfinity

What's New *1*

Proposal fully implemented in reference interpreter

- can validate, execute and inter-convert wa(s)t/wasm
- based on typed function proposal
- should match V8 prototype

What's New *2*

Basic test suite

- covers all MVP constructs, but could use more tests
- contributions welcome!

What's New *3*

Wrote a compiler targeting the proposal

Introducing **Wob**

A mini OO language in the style of C# and friends

Meant to be representative wrt main challenges

Main Challenges in targeting Wasm GC

- ✓ Generics
- (✓) Classes
- User-level casts
- ✓ Separate compilation

for MVP, expressiveness matters more than performance

Wob Features

primitive data types

tuples and arrays

functions and closures

classes and inheritance

generics

safe downcasts

modules with separate compilation and client-side linking

Wob Implementation

Interpreter

Compiler to Wasm

REPL that can evaluate through Wasm

Compiled Wasm is self-contained

On **wob** branch in GC repo

Wob Demo

Pain Points so far

Lots of type annotations

ref.as_* instructions trap on null

No freeze/readonly mode (need 2-phase instance alloc)

Non-nullable types mostly useless without locals

No bulk instructions on arrays

No non-trivial const expressions (i31.new, rtt.canon, …)

Lack of null type

What worked

Generics over boxed types

Classes

Separate compilation (tuples and arrays are structural)

Writing a compiler against Wasm is a luxury

- validation catches most representation bugs
- …and immediately points to broken piece of codegen

Wob Todos

Compile overriding

Compile downcasts

Exercise multiple impl strategies for generics

Exercise multiple impl strategies for classes

Add non-null types and exercise non-null refs

Summary

Compilation Scheme

Source Type	as value	as field	boxed
Bool	i32	i8	i31ref
Byte	i32	i8	i31ref
Int	i32		ref (struct i32)
Float	f64		ref (struct f64)
Text	ref (array i8)	anyref	
(Float, Text)	ref (struct f64 anyref)	anyref	
Float[]	ref (array f64)	anyref	
Text[]	ref (array anyref)	anyref	
C	ref (struct (ref $Cvt) …)	anyref	
<T>	anyref		

Classes

Class is represented by 6-tuple:

- dispatch table struct
- RTT
- new function
- pre-alloc function (called by subclasses’ new/pre-alloc function)
- post-alloc function (called by subclasses’ new/post-alloc function)
- superclass

2-phase init to handle immutable fields and subtyping

- first phase evaluates args and immutable initialisers
- second phase evaluates exps and mutable initialisers
- can observe uninitialised mutable bindings but not immutable

Approaches to Generics

1. C++ style: static type specialisation
   (too restrictive for HL langs, cannot handle generic methods, closures, etc.)

2. C# style: dynamic type specialisation
   (not easy in Wasm, need two-level jit & dynamic linking)

3. Java style: generics only allowed for boxed types
   (simple but limited, requires downcasts on generic results)

4. Functional language style: universal representation
   (requires more representation changes)

5. Dynamic language style: dispatch on type
   (generic access becomes much more expensive)

Wob currently does 3,
want to try 4 and 5 as well

Casts

Not done yet

Generic type parameters need to be reified to values in user space

Wasm-level RTTs and casts are insufficient for this, since they cannot represent generic types

Until Wasm has generics, you have to roll your own