Staged Compilation through Module Linking

Andreas Rossberg

Staged Compilation

Wasm Compilation Model

compilation of a module happens before linking

linking happens through instantiation, providing imports

compiled module can be instantiated multiple times

… linking the same code against different imports

hence, compiler cannot know imports

Use Cases for Compile-time Imports

system libraries

… provide efficient access to host-specific functionality, avoid feature creep in Wasm

… e.g., DOM access, string library, cutting edge SIMD :), low-level engine primitives

… Wasm compiler can recognise and optimise host primitives

type specialisation

… import types without uniform representation

… compiler can produce heterogeneous code

cross-module optimisation

… compiler can inline functions, constant folding,
perform other value-dependent optimisations involving imports

Staged Instantiation

naive idea: allow both compile-time and link-time imports

a module can mark selective imports as pre-imports

- … have to be provided at compile time
- … must not depend on regular imports

Pre-imports

```
(module $regex

    (preimport "sys" "string" (type $string))
    (preimport "sys" "get" (func $get (param (ref $string) i32) (result i32)))

    (import "io" "open" (func … ))

    (func $match (param $re (ref $string)) (result i32)
        …
        (call $get (local.get $re) (local.get $i))
        …
    )
)
```

compiler knows $get function and can compile call into direct code, IC, etc.

Pre-imports

```
(module $regex

    (preimport "string" "string" (type $string))
    (preimport "string" "get" (func $get (param (ref $string) i32) (result i32)))

    (import "io" "open" (func … ))

    (func $matcher (param $re (ref $string)) (result i32)
        …
        (call $get (local.get $re) (local.get $i))
        …
    )
)

let regexModule = new Module(regexBinary, {"string" : StringLib})
post(regexModule)
— — — — — — — — —
let regex = new Instance(regexModule, {"io" : LocalIoLib})
```

Staged Compilation

yields a system for staged compilation

what if a module's pre-import needs to be supplied by another's export?

… do we need pre-exports, too?

… ultimately, a "pre" version of everything?

what if somebody needs more than 2 stages?

… staged languages typically support arbitrary tower

module linking to the rescue…

Module Linking

Module Linking Recap

was separate from component proposal before

extend core Wasm with nested instances and modules

desirable for various reasons

- … standardised language for describing Wasm linking
- … self-contained model of deployment for complex apps
- … parameterised imports

Module Linking Syntax Recap

nested instance and module definitions resp. sections

```
(module $m (import “a” “b” (func)) …)
(instance $i (instantiate $m (with “a” “b” (func $f))))
```

instance and module imports and exports

```
(export “instance” (instance $i))
(import “a” “m” (module $M))
```

instance and module types

```
(type $M (module
  (import “a” “f” (func (param i32)))
  (export “g” (func (result i32)))
))
```

auxiliary alias declarations to access out index spaces

Staging via Module Linking

key assumption

nested modules are compiled when enclosing module is instantiated

hence, when compiled, outer imports are already known

… nested modules are “closures” wrt outer scope

composable, as it can be repeated at arbitrary nesting depth

Staging with Pre-imports

```
(module $regex

    (preimport "sys" "string" (type $string))
    (preimport "sys" "get" (func $get (param (ref $string) i32) (result i32)))

    (import "io" "open" (func … ))

    (func $match (param $re (ref $string)) (result i32)
        …
        (call $get (local.get $re) (local.get $i))
        …
    )
)
```

Staging with Nested Module

```
(module $regex

  (import "string" "string" (type $string))
  (import "string" "get" (func $get (param (ref $string) i32) (result i32)))

  (module $main (export "Main")
    (import "io" "open" (func … ))

    (func $match (param $re (ref $string)) (result i32)
      …
      (call $get (local.get $re) (local.get $i))
      …
    )
  )
)
```

Staging with Nested Module

```
let regexModule = new Module(regexBinary)
let regexPre = new Instance(regexModule,{"string" : StringLib})
post(regexPre.exports.Main)
— — — — — — — — —
let regex = new Instance(regexModule, {"io" : LocalIoLib})
```

Summary

Composable way of expressing staging

Additional use case for module linking

Worth reviving module linking proposal?