Reflecting Types in the Wasm JS API

Proposal

Andreas Rossberg

Dfinity

Motivation

Wasm is typed

Types carry useful information

…form of imports and exports
…sizes of tables and memories
…mutability and contents of globals
…signature of functions

Desire to query this information came up repeatedly

…e.g., JS-hosted linker, module adaptors, etc.

Proposal Summary

Systematic representation of Wasm types as JS objects

Extend API classes with `.type` methods

Adapt constructors to accept types

Add `WebAssembly.Function` class

Wasm Types as JSON

```
type ValueType = "i32" | "i64" | "f32" | "f64"
type ElemType = "anyfunc"

type FunctionType = {params: ValueType[], results: ValueType[]}
type GlobalType = {value: ValueType, mutable: bool}
type MemoryType = {limits: Limits}
type TableType = {limits: Limits, element: ElemType}
type Limits = {min: num, max?: num}

type ExternType =
  {kind: "function", type: FunctionType} |
  {kind: "memory", type: MemoryType} |
  {kind: "table", type: TableType} |
  {kind: "global", type: GlobalType}
```

Wasm Types vs JS API

Current API “descriptor” interfaces mostly match Wasm types already

Some naming differences
…mostly spec-internal, rename for clarity (`*Descriptor` to `*Type`)

A couple of missing interfaces
…add `FunctionType`, `ExternType`

Minor structural differences
…`Import/ExportDescriptor` contain names
(make them subinterfaces inheriting from `ExternType`)

…limits are inlined
(cosmetic difference, does not matter)

API Extensions

Static methods to retrieve types

```
Memory.type(Memory) : MemoryType
Table.type(Table) : TableType
Global.type(Global) : GlobalType
```

Inversely, constructors accept types

```
new Memory(MemoryType)
new Table(TableType)
new Global(GlobalType, value)
```

```
function mockImports(module) {
  let mock = {};

  for (let import of WebAssembly.Module.imports(module)) {
    let value;
    switch (import.kind) {
      case "table":
        value = new WebAssembly.Table(import.type); break;
      case "memory":
        value = new WebAssembly.Memory(import.type); break;
      case "global":
        value = new WebAssembly.Global(import.type, undefined); break;
      case "function":
        value = function () { throw "unimplemented" }; break;
    }

    if (! (import.module in mock)) mock[import.module] = {};
    mock[import.module][import.name] = value;
  }
  return mock;
}

let module = ...;
let instance = WebAssembly.instantiate(module, mockImports(module));
```

WebAssembly.Function

Class of “Wasm exported function objects”

…subclass of JS’s `Function`

Analogous to other API classes

`Function.type(Function) : FunctionType`

`new Function(FunctionType, function)`

…creates “Wasm exported function object” from any JS function

…closing an existing gap in API

…providing a way to store JS functions in tables

…and passing them as anyfunc arguments!

```
function print(...args) {
  for (let x of args) console.log(x + "\n")
}

let table = new Table(
    {element: "anyfunc", initial: 10});

let print_i32 = new WebAssembly.Function(
    {parameters: ["i32"], results: []}, print);

table.set(0, print_i32);
```

Wrinkle: naming of limits

API calls lower and upper size `initial` and `maximum`

Fitting for constructors, but not for general use as type

…lower also reflects current size (in result of .type)

…or required size (in import descriptions)

…consequently, Wasm symmetrically calls it `minimum`

Should rename!

…for backwards compatibility,
keep accepting old name in constructors (overload)