Please

Ask questions through the app

Rate Session

Thank you!

@GOTOber

gotober.com

WebAssembly
what? why? whither?

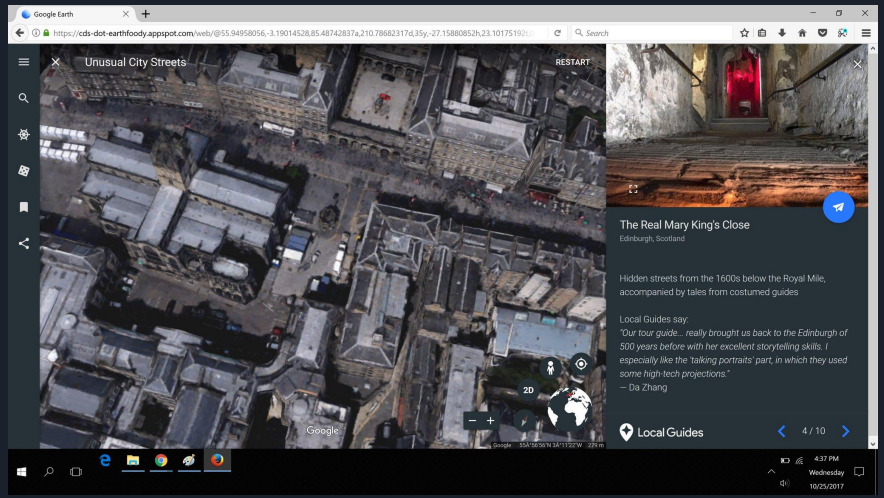
Ben L. Titzer
Andreas Rossberg
Google Germany

What is WebAssembly?

- A portable, compact, binary code format
  - Low-level execution model with native unboxed types
  - Suitable as a compilation target, e.g. for C++
- Natively supported in all major browsers
- Developed in the W3C WebAssembly Community Group
- Offers near-native performance for low-level code
- Integrates with the Web platform through JavaScript APIs

Users and potential users

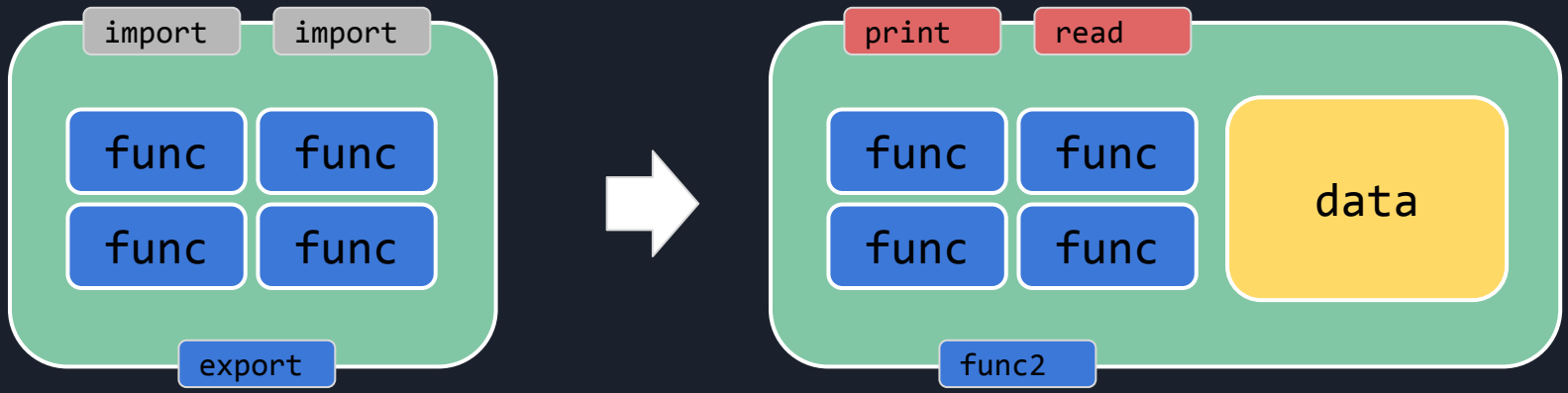
- WebAssembly has grown fast since first browsers shipped in March 2017, with many prototypes and a lot of interest
  - Google Earth
  - Unity3d
  - Epic
  - AutoCAD
  - Figma
  - Farmville 2
  - Active Financial
  - Soundation
  - AmpedStudio
  - Twitch.tv



WebAssembly Basics - Modules

- WebAssembly code is organized into units called *modules*
- A module has a number of *functions*
  - Imports: for accessing explicitly granted host functionality
  - Locally defined functions: local computation
  - Exported functions: allows calls into the module
- A module has no *state*, but declares:
  - Size of memory
  - Global variables
  - Tables for indirect calls and host interaction
- A module is *instantiated* to create a stateful computation
  - An instance has a memory, variables, local execution stack, etc

WebAssembly Modules and Instances



Module

Instance

Binary Format Overview

- Design:
  - A series of *sections* in a pre-defined order
  - Use of variable-length integers for future-proofing
  - Function body code is stack machine *bytecode*

Binary Format

Function signatures	`code=0x01`
Imports	`code=0x02`
Function declarations	`code=0x03`
Indirect table	`code=0x04`
Memory configuration	`code=0x05`
Global declarations	`code=0x06`
Exports	`code=0x07`
Start function index	`code=0x08`
Table initialization section	`code=0x09`
Function bodies (code)	`code=0x0a`
Memory initialization data	`code=0x0b`

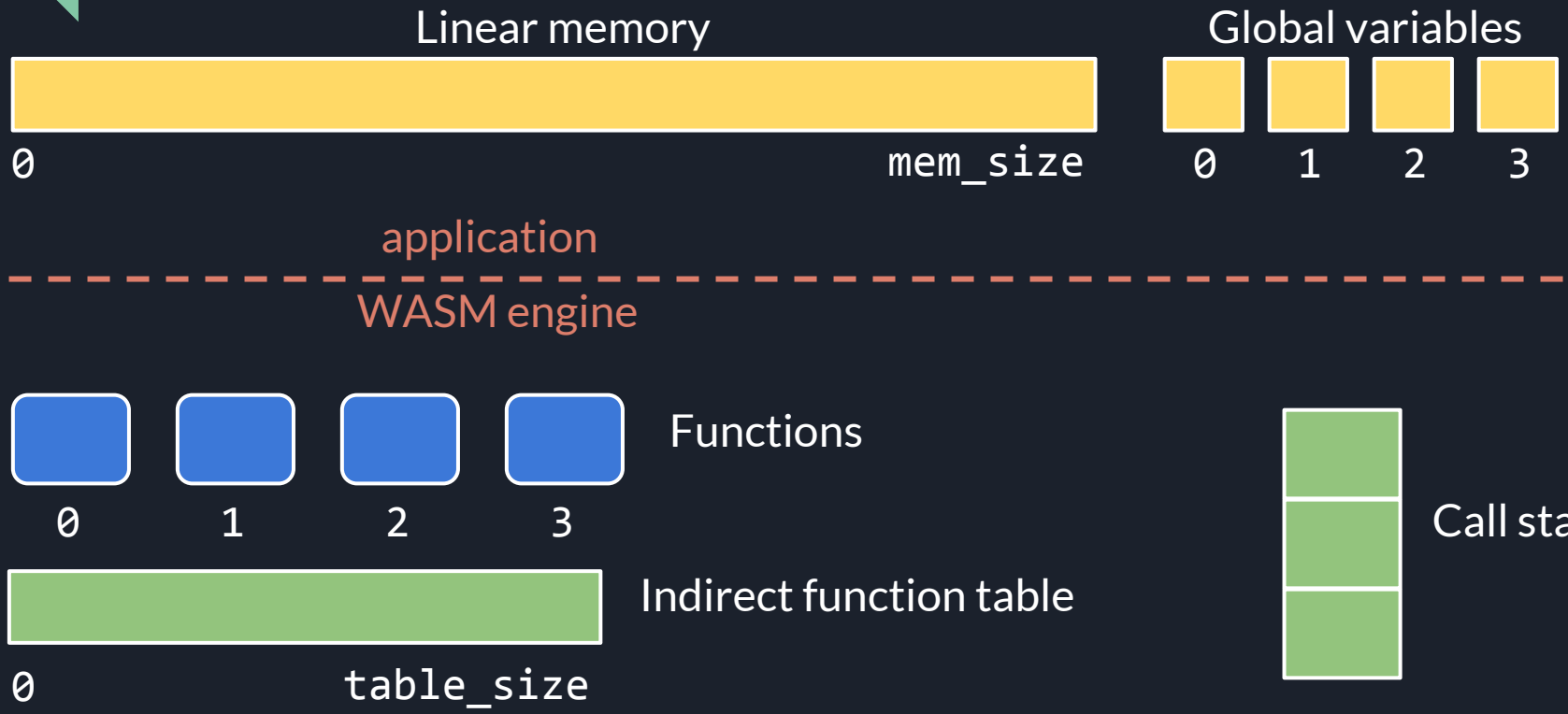
Binary Format proportions

First sections

Function bodies (code) ~90%

Memory initialization data

Execution Model



(Simplified) Instruction Set

Types

i32 i64 f32 f64

Instructions

i32.add	i64.add	f32.add	f64.add	call	block
i32.sub	i64.sub	f32.sub	f64.sub	call_indirect	if
i32.lt	i64.lt			return	loop
					br

get_local	load_global	i32.load_mem
set_local	store_global	i32.store_mem

Instruction Example

Official text format

```
(func (export "add")
    (param $x i32)
    (param $y i32)
    (result i32)
  (i32.add
      (get_local $x)
      (get_local $y)))
```

Binary

```
+15: 60 02 7f 7f 01 7f

. . .

+27: 01 00

. . .

+48: 01 87 80 80 80 00
+54: 00 20 00 20 01 6a 0b
```

Types section

```
(int, int) -> int
```

Function declaration

```
1 function of type #0
```

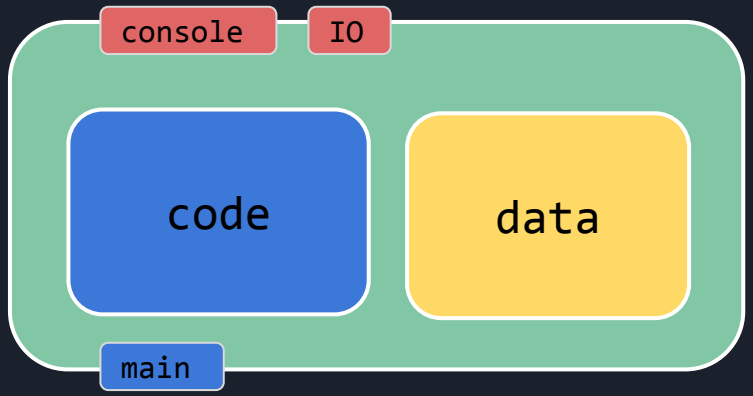
Function body

```
1 body of length 7
0 additional locals
get_local #0
get_local #1
i32.add
end
```

Embedding WebAssembly

- WebAssembly is not a complete platform, but is more like a virtual instruction set architecture, like X86
- Requires an *embedding* environment to provide:
  - Module loading and instantiation
  - Linking between modules (via imports and exports)
  - Host imported functions for interacting with the platform
- Most important embedding is within JavaScript, within the web
  - APIs for loading and instantiating modules
  - Provides access to JavaScript APIs
  - Callable from JavaScript
  - Offers no new “API surface” for the web

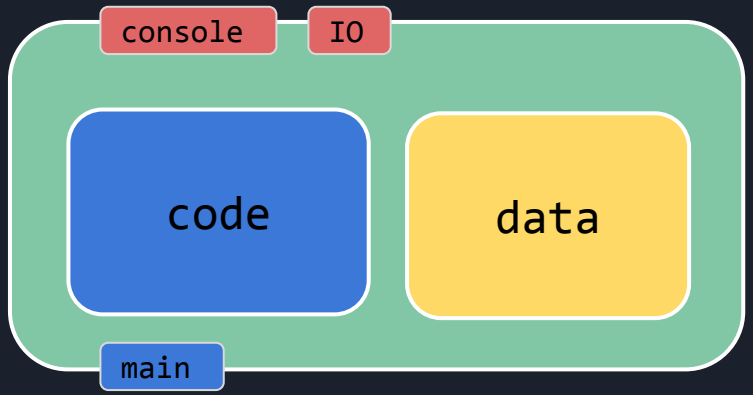
Embedding WebAssembly



- WebAssembly instances can only interact with APIs that are provided by an embedder

Explicitly imported JavaScript functions

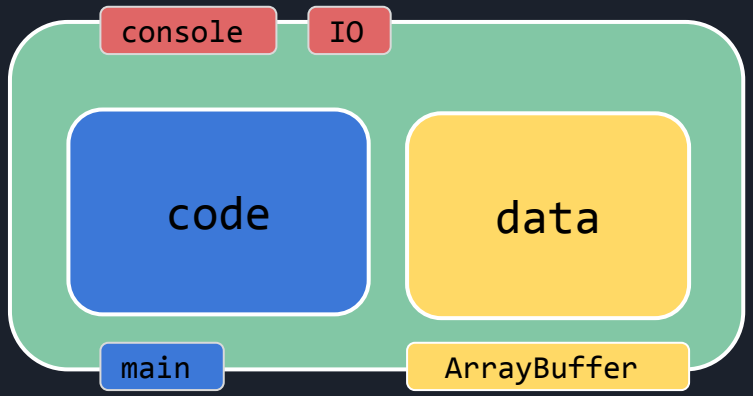
JavaScript API to load, validate, and instantiate modules



Exported functions are called as normal JavaScript functions

Explicitly imported JavaScript functions

JavaScript API to load, validate, and instantiate modules



Exported functions are called as normal JavaScript functions

The memory can optionally be exported to JS as an array buffer

JavaScript API

- New `WebAssembly` object available in JavaScript outermost scope
  - `WebAssembly.Module` - opaque representation of decoded binary bytes
  - `WebAssembly.Instance` - instantiated module with state
  - `WebAssembly.Memory` - handle to low-level memory
  - `WebAssembly.Table` - used for indirect calls
- Synchronous and asynchronous API for compiling modules
  - A simple as `new WebAssembly.Module(bytes)` and `new WebAssembly.Instance(module, imports)`

Why WebAssembly?

High performance

Predictable performance

Empower other languages than JavaScript

Enable features that JavaScript can’t provide

Supersede asm.js and (P)NaCl

Goals & Constraints

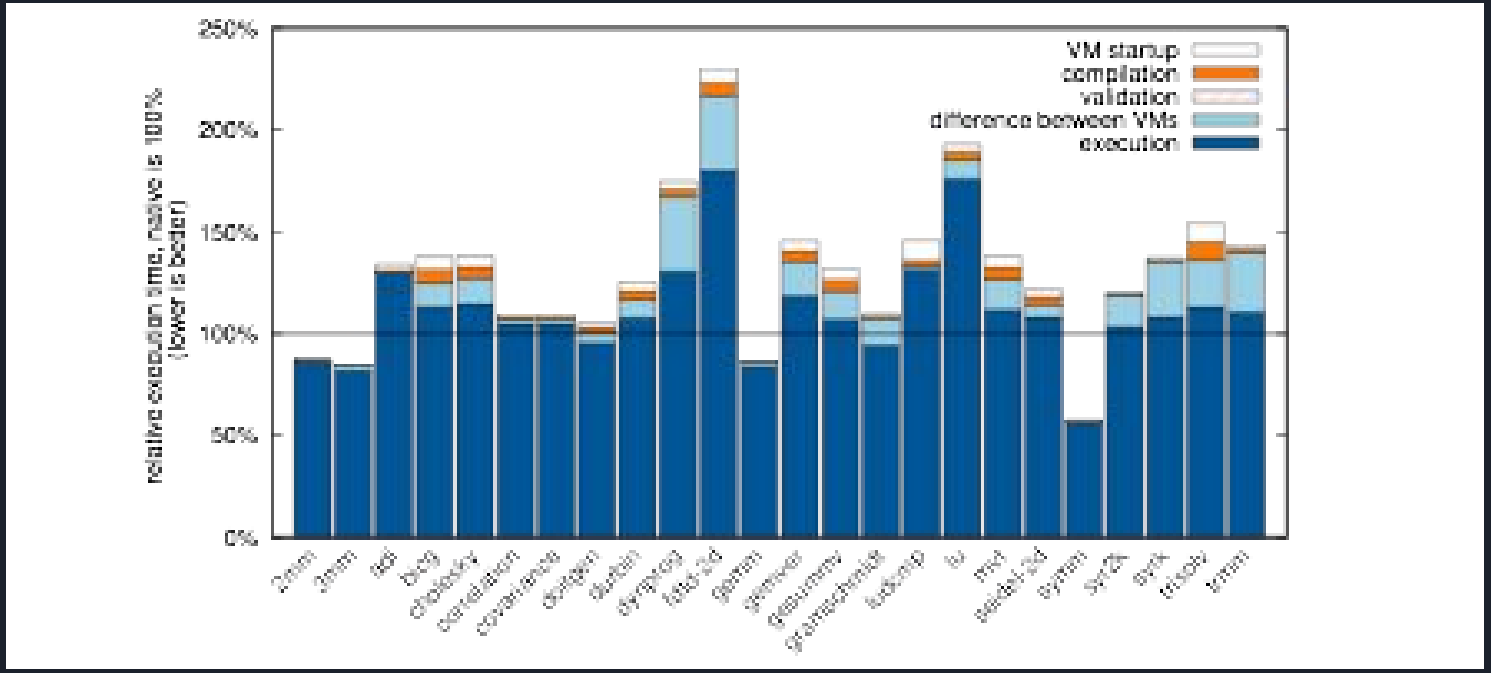
Semantics

- Language-independent
- Platform-independent
- Hardware-independent
- Fast to execute
- Safe to execute
- Deterministic
- Easy to reason about

Representation

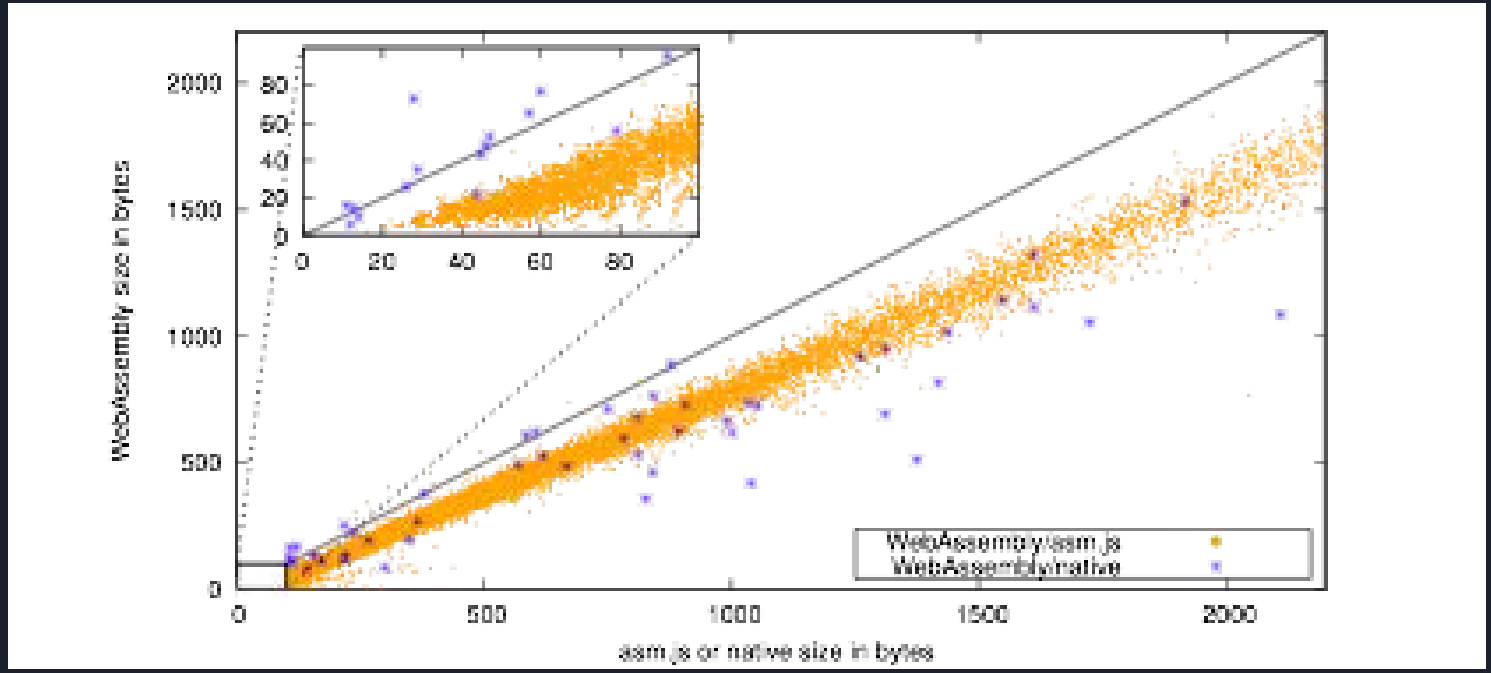
- Compact
- Easy to generate
- Fast to decode
- Fast to validate
- Fast to compile
- Streamable
- Parallelizable

Performance



[PLDI 2017]

Code Size



[PLDI 2017]

Implementation

WebAssembly is implemented *within* JavaScript engine

- Reuses sophisticated optimising JIT
- Fast calls from Wasm to JS and vice versa
- No need to reinvent garbage collection, code management, etc

Different implementation strategies

- Multiple tiers
- AOT compilation, JIT compilation, or interpretation as first tier

Additional optimisations

- Code specialisation, hardware traps, etc

Producing WebAssembly

Compiling from different languages

- C and C++ (via Emscripten)
- Rust
- AssemblyScript
- ... hopefully many more to come soon

Generating Wasm binaries at runtime

- Can implement source language JITs on top of Wasm

Designed to be language-independent!

Consuming WebAssembly

In the browser
- As part of the web platform
- Embedded into JavaScript

Other embeddings and standalone implementations
- Many potential use cases independent of the web or JavaScript
- Some already being build
- Stay tuned.

Designed to be platform/embedding-independent!

The Specification

Contract between producers and consumers

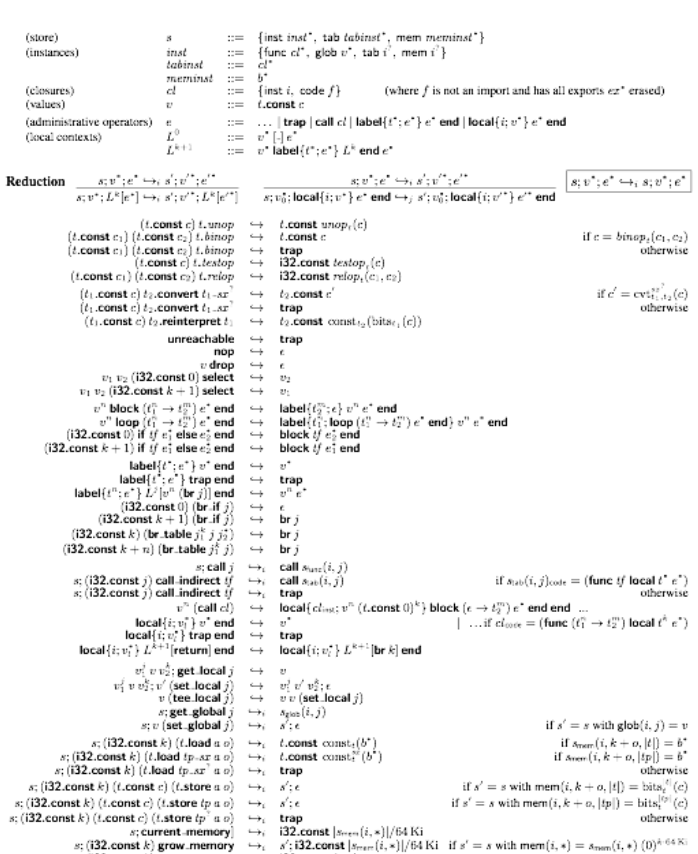
Setting a new bar for industrial languages!

Includes a complete formal semantics

- Declarative specification of binary format, validation, execution
- Using state-of-the-art techniques from research
- Led to simpler and cleaner design

Mechanised and machine-verified in theorem prover

- Proof of type soundness
- Absence of undefined behaviour, sandbox safety



(store)	s	::=	$\{\text{inst } inst^*,\ \text{tab } tabinst^*,\ \text{mem } meminst^*\}$
(instances)	$inst$	::=	$\{\text{func } cl^*,\ \text{glob } v^*,\ \text{tab } i^?,\ \text{mem } i^?\}$
	$tabinst$	::=	cl^*
	$meminst$	::=	b^*
(closures)	cl	::=	$\{\text{inst } i,\ \text{code } f\}$ (where f is not an import and has all exports ex^* erased)
(values)	v	::=	$t.\text{const } c$
(administrative operators)	e	::=	$\ldots \mid \textbf{trap} \mid \textbf{call } cl \mid \textbf{label}\{t^*; e^*\}\ e^*\ \textbf{end} \mid \textbf{local}\{i; v^*\}\ e^*\ \textbf{end}$
(local contexts)	L^0	::=	$v^*\ [\cdot]\ e^*$
	L^{k+1}	::=	$v^*\ \textbf{label}\{t^*; e^*\}\ L^k\ \textbf{end}\ e^*$

Reduction

$$\frac{s; v^*; e^* \hookrightarrow_i s'; v'^*; e'^*}{s; v^*; L^k[e^*] \hookrightarrow_i s'; v'^*; L^k[e'^*]} \qquad \frac{s; v^*; e^* \hookrightarrow_i s'; v'^*; e'^*}{s; v_0^*; \textbf{local}\{i; v^*\}\ e^*\ \textbf{end} \hookrightarrow_j s'; v_0^*; \textbf{local}\{i; v'^*\}\ e'^*\ \textbf{end}} \qquad \boxed{s; v^*; e^* \hookrightarrow_i s; v^*; e^*}$$

$$
\begin{array}{rcll}
(t.\text{const } c_1)\ (t.\text{const } c_2)\ t.binop & \hookrightarrow & t.\text{const } c & \text{if } c = binop_t(c_1, c_2) \\
(t.\text{const } c_1)\ (t.\text{const } c_2)\ t.binop & \hookrightarrow & \textbf{trap} & \text{otherwise} \\
(t.\text{const } c)\ t.testop & \hookrightarrow & \text{i32.const } testop_t(c) & \\
(t.\text{const } c_1)\ (t.\text{const } c_2)\ t.relop & \hookrightarrow & \text{i32.const } relop_t(c_1, c_2) & \\
(t_1.\text{const } c)\ t_2.\textbf{convert } t_1.sx^? & \hookrightarrow & t_2.\text{const } c' & \text{if } c' = \text{cvt}^{sx^?}_{t_1,t_2}(c) \\
(t_1.\text{const } c)\ t_2.\textbf{convert } t_1.sx^? & \hookrightarrow & \textbf{trap} & \text{otherwise} \\
(t_1.\text{const } c)\ t_2.\textbf{reinterpret } t_1 & \hookrightarrow & t_2.\text{const } \text{const}_{t_2}(\text{bits}_{t_1}(c)) & \\
\textbf{unreachable} & \hookrightarrow & \textbf{trap} & \\
\textbf{nop} & \hookrightarrow & \epsilon & \\
v\ \textbf{drop} & \hookrightarrow & \epsilon & \\
v_1\ v_2\ (\text{i32.const } 0)\ \textbf{select} & \hookrightarrow & v_2 & \\
v_1\ v_2\ (\text{i32.const } k+1)\ \textbf{select} & \hookrightarrow & v_1 & \\
v^n\ \textbf{block}\ \{t_1^n \to t_2^m\}\ e^*\ \textbf{end} & \hookrightarrow & \textbf{label}\{t_2^m; \epsilon\}\ v^n\ e^*\ \textbf{end} & \\
v^n\ \textbf{loop}\ \{t_1^n \to t_2^m\}\ e^*\ \textbf{end} & \hookrightarrow & \textbf{label}\{t_1^n; \textbf{loop}\ \{t_1^n \to t_2^m\}\ e^*\ \textbf{end}\}\ v^n\ e^*\ \textbf{end} & \\
(\text{i32.const } 0)\ \textbf{if } tf\ e_1^*\ \textbf{else } e_2^*\ \textbf{end} & \hookrightarrow & \textbf{block } tf\ e_2^*\ \textbf{end} & \\
(\text{i32.const } k+1)\ \textbf{if } tf\ e_1^*\ \textbf{else } e_2^*\ \textbf{end} & \hookrightarrow & \textbf{block } tf\ e_1^*\ \textbf{end} & \\
\textbf{label}\{t^*; e^*\}\ v^*\ \textbf{end} & \hookrightarrow & v^* & \\
\textbf{label}\{t^*; e^*\}\ \textbf{trap end} & \hookrightarrow & \textbf{trap} & \\
\textbf{label}\{t^n; e^*\}\ L^j[v^n\ (\textbf{br } j)]\ \textbf{end} & \hookrightarrow & v^n\ e^* & \\
(\text{i32.const } 0)\ (\textbf{br\_if } j) & \hookrightarrow & \epsilon & \\
(\text{i32.const } k+1)\ (\textbf{br\_if } j) & \hookrightarrow & \textbf{br } j & \\
(\text{i32.const } k)\ (\textbf{br\_table } j^?\ j_0^*\ j) & \hookrightarrow & \textbf{br } j & \\
(\text{i32.const } k+n)\ (\textbf{br\_table } j_1^k\ j\ j_2^*) & \hookrightarrow & \textbf{br } j & \\
s; \textbf{call } j & \hookrightarrow_i & \textbf{call } s_{\text{inst}}(i, j) & \\
s; (\text{i32.const } j)\ \textbf{call\_indirect } tf & \hookrightarrow_i & \textbf{call } s_{\text{tab}}(i, j) & \text{if } s_{\text{tab}}(i, j)_{\text{code}} = (\textbf{func } tf\ \textbf{local } t^*\ e^*) \\
s; (\text{i32.const } j)\ \textbf{call\_indirect } tf & \hookrightarrow_i & \textbf{trap} & \text{otherwise} \\
v^n\ (\textbf{call } cl) & \hookrightarrow & \textbf{local}\{cl_{\text{inst}}; v^n\ (t.\text{const } 0)^k\}\ \textbf{block}\ (\epsilon \to t_2^m)\ e^*\ \textbf{end end} \quad \ldots & \\
\textbf{local}\{i; v_1^n\}\ v^n\ \textbf{end} & \hookrightarrow & v^* \quad \mid \quad \ldots \text{if } cl_{\text{code}} = (\textbf{func}\ \{t_1^n \to t_2^m\}\ \textbf{local}\ t^k\ e^*) & \\
\textbf{local}\{i; v_1^*\}\ \textbf{trap end} & \hookrightarrow & \textbf{trap} & \\
\textbf{local}\{i; v_1^*\}\ L^{k+1}[\textbf{return}]\ \textbf{end} & \hookrightarrow & \textbf{local}\{i; v_1^*\}\ L^{k+1}[\textbf{br } k]\ \textbf{end} & \\
v_1^i\ v\ v_2^k; \textbf{get\_local } j & \hookrightarrow & v & \\
v_1^i\ v\ v_2^k; v'\ (\textbf{set\_local } j) & \hookrightarrow & v_1^i\ v'\ v_2^k; \epsilon & \\
v\ (\textbf{tee\_local } j) & \hookrightarrow & v\ v\ (\textbf{set\_local } j) & \\
s; \textbf{get\_global } j & \hookrightarrow_i & s_{\text{glob}}(i, j) & \\
s; v\ (\textbf{set\_global } j) & \hookrightarrow_i & s'; \epsilon & \text{if } s' = s \text{ with } \text{glob}(i, j) = v \\
s; (\text{i32.const } k)\ (t.\textbf{load a } o) & \hookrightarrow_i & t.\text{const } \text{const}_t(b^*) & \text{if } s_{\text{mem}}(i, k+o, |t|) = b^* \\
s; (\text{i32.const } k)\ (t.\textbf{load } tp\_sx\ a\ o) & \hookrightarrow_i & t.\text{const } \text{const}_t^{sx}(b^*) & \text{if } s_{\text{mem}}(i, k+o, |tp|) = b^* \\
s; (\text{i32.const } k)\ (t.\textbf{load } tp\_sx^?\ a\ o) & \hookrightarrow_i & \textbf{trap} & \text{otherwise} \\
s; (\text{i32.const } k)\ (t.\text{const } c)\ (t.\textbf{store a } o) & \hookrightarrow_i & s'; \epsilon & \text{if } s' = s \text{ with } \text{mem}(i, k+o, |t|) = \text{bits}_t^{?}(c) \\
s; (\text{i32.const } k)\ (t.\text{const } c)\ (t.\textbf{store } tp\ a\ o) & \hookrightarrow_i & s'; \epsilon & \text{if } s' = s \text{ with } \text{mem}(i, k+o, |tp|) = \text{bits}_t^{|tp|}(c) \\
s; (\text{i32.const } k)\ (t.\text{const } c)\ (t.\textbf{store } tp^?\ a\ o) & \hookrightarrow_i & \textbf{trap} & \text{otherwise} \\
s; \textbf{current\_memory} & \hookrightarrow_i & \text{i32.const } |s_{\text{mem}}(i, *)|/64\,\text{Ki} & \\
s; (\text{i32.const } k)\ \textbf{grow\_memory} & \hookrightarrow_i & s'; \text{i32.const } |s_{\text{mem}}(i, *)|/64\,\text{Ki} & \text{if } s' = s \text{ with } \text{mem}(i, *) = s_{\text{mem}}(i, *)\ (0)^{k \cdot 64\,\text{Ki}} \\
s; (\text{i32.const } k)\ \textbf{grow\_memory} & \hookrightarrow_i & \text{i32.const } (-1) &
\end{array}
$$

Figure 1. Small-step reduction rules

[PLDI 2017]

$$\text{(contexts)}\quad C \;::=\; \{\text{func } tf^*,\ \text{global } tg^*,\ \text{table } n^?,\ \text{memory } n^?,\ \text{local } t^*,\ \text{label } (t^*)^*\}$$

Typing Instructions $\boxed{C \vdash e^* : tf}$

$$\frac{}{C \vdash t.\mathbf{const}\ c : \epsilon \to t} \qquad \frac{}{C \vdash t.unop : t \to t} \qquad \frac{}{C \vdash t.binop : t\ t \to t} \qquad \frac{}{C \vdash t.testop : t \to \mathbf{i32}} \qquad \frac{}{C \vdash t.relop : t\ t \to \mathbf{i32}}$$

$$\frac{t_2 \neq t_2 \qquad sx^? = \epsilon \Leftrightarrow (t_1 = \mathbf{in} \wedge t_2 = \mathbf{in}' \wedge |t_1| < |t_2|) \vee (t_1 = \mathbf{fn} \wedge t_2 = \mathbf{fn}')}{C \vdash t_1.\mathbf{convert}\ t_2.sx^? : t_2 \to t_1} \qquad \frac{t_1 \neq t_2 \qquad |t_1| = |t_2|}{C \vdash t_1.\mathbf{reinterpret}\ t_2 : t_2 \to t_1}$$

$$\frac{}{C \vdash \mathbf{unreachable} : t_1^* \to t_2^*} \qquad \frac{}{C \vdash \mathbf{nop} : \epsilon \to \epsilon} \qquad \frac{}{C \vdash \mathbf{drop} : t \to \epsilon} \qquad \frac{}{C \vdash \mathbf{select} : t\ t\ \mathbf{i32} \to t}$$

$$\frac{tf = t_1^n \to t_2^m \qquad C, \mathbf{label}\ (t_2^m) \vdash e^* : tf}{C \vdash \mathbf{block}\ tf\ e^*\ \mathbf{end} : tf} \qquad \frac{tf = t_1^n \to t_2^m \qquad C, \mathbf{label}\ (t_1^n) \vdash e^* : tf}{C \vdash \mathbf{loop}\ tf\ e^*\ \mathbf{end} : tf}$$

$$\frac{tf = t_1^n \to t_2^m \qquad C, \mathbf{label}\ (t_2^m) \vdash e_1^* : tf \qquad C, \mathbf{label}\ (t_2^m) \vdash e_2^* : tf}{C \vdash \mathbf{if}\ tf\ e_1^*\ \mathbf{else}\ e_2^*\ \mathbf{end} : t_1^n\ \mathbf{i32} \to t_2^m}$$

$$\frac{C_{\mathbf{label}}(i) = t^*}{C \vdash \mathbf{br}\ i : t_1^*\ t^* \to t_2^*} \qquad \frac{C_{\mathbf{label}}(i) = t^*}{C \vdash \mathbf{br\_if}\ i : t^*\ \mathbf{i32} \to t^*} \qquad \frac{(C_{\mathbf{label}}(i) = t^*)^+}{C \vdash \mathbf{br\_table}\ i^+ : t_1^*\ t^*\ \mathbf{i32} \to t_2^*}$$

$$\frac{C_{\mathbf{label}}(|C_{\mathbf{label}}| - 1) = t^*}{C \vdash \mathbf{return} : t_1^*\ t^* \to t_2^*} \qquad \frac{C_{\mathbf{func}}(i) = tf}{C \vdash \mathbf{call}\ i : tf} \qquad \frac{tf = t_1^* \to t_2^* \qquad C_{\mathbf{table}} = n}{C \vdash \mathbf{call\_indirect}\ tf : t_1^*\ \mathbf{i32} \to t_2^*}$$

$$\frac{C_{\mathbf{local}}(i) = t}{C \vdash \mathbf{get\_local}\ i : \epsilon \to t} \qquad \frac{C_{\mathbf{local}}(i) = t}{C \vdash \mathbf{set\_local}\ i : t \to \epsilon} \qquad \frac{C_{\mathbf{local}}(i) = t}{C \vdash \mathbf{tee\_local}\ i : t \to t} \qquad \frac{C_{\mathbf{global}}(i) = \mathbf{mut}^?\ t}{C \vdash \mathbf{get\_global}\ i : \epsilon \to t} \qquad \frac{C_{\mathbf{global}}(i) = \mathbf{mut}\ t}{C \vdash \mathbf{set\_global}\ i : t \to \epsilon}$$

$$\frac{C_{\mathbf{memory}} = n \qquad 2^n < (|tp| <)^?|t| \qquad (tp.sx)^? = \epsilon \vee t = \mathbf{in}}{C \vdash t.\mathbf{load}\ (tp.sx)^?\ a\ o : \mathbf{i32} \to t} \qquad \frac{C_{\mathbf{memory}} = n \qquad 2^n \leq (|tp| <)^?|t| \qquad tp^? = \epsilon \vee t = \mathbf{in}}{C \vdash t.\mathbf{store}\ tp^?\ a\ o : \mathbf{i32}\ t \to \epsilon}$$

$$\frac{C_{\mathbf{memory}} = n}{C \vdash \mathbf{current\_memory} : \epsilon \to \mathbf{i32}} \qquad \frac{C_{\mathbf{memory}} = n}{C \vdash \mathbf{grow\_memory} : \mathbf{i32} \to \mathbf{i32}}$$

$$\frac{}{C \vdash \epsilon : \epsilon \to \epsilon} \qquad \frac{C \vdash e_1^* : t_1^* \to t_2^* \qquad C \vdash e_2 : t_2^* \to t_3^*}{C \vdash e_1^*\ e_2 : t_1^* \to t_3^*} \qquad \frac{C \vdash e^* : t_1^* \to t_2^*}{C \vdash e^* : t^*\ t_1^* \to t^*\ t_2^*}$$

Typing Modules

$$\frac{tf = t_1^* \to t_2^* \qquad C, \mathbf{local}\ t_1^*\ t^*, \mathbf{label}\ (t_2^*) \vdash e^* : \epsilon \to t_2^*}{C \vdash ex^*\ \mathbf{func}\ tf\ \mathbf{local}\ t^*\ e^* : ex^*\ tf} \qquad \frac{tg = \mathbf{mut}^?\ t \qquad C \vdash e^* : \epsilon \to t \qquad ex^* = \epsilon \vee tg = t}{C \vdash ex^*\ \mathbf{global}\ tg\ e^* : ex^*\ tg}$$

$$\frac{(C_{\mathbf{func}}(i) = tf)^n}{C \vdash ex^*\ \mathbf{table}\ n\ i^n : ex^*\ n} \qquad \frac{}{C \vdash ex^*\ \mathbf{memory}\ n : ex^*\ n}$$

$$\frac{tg = t}{C \vdash ex^*\ \mathbf{func}\ tf\ im : ex^*\ tf} \qquad \frac{}{C \vdash ex^*\ \mathbf{global}\ tg\ im : ex^*\ tg} \qquad \frac{}{C \vdash ex^*\ \mathbf{table}\ n\ im : ex^*\ n} \qquad \frac{}{C \vdash ex^*\ \mathbf{memory}\ n\ im : ex^*\ n}$$

$$\frac{\begin{array}{c}(C \vdash f : ex_1^*\ tf)^* \qquad (C_i \vdash glob_i : ex_0^*\ tg_i)_i^* \qquad (C \vdash tab : ex_1^*\ n)^? \qquad (C \vdash mem : ex_0^*\ n)^? \\ C_i = \{\mathbf{global}\ tg^{i-1}\}_i^* \qquad C = \{\mathbf{func}\ tf^*, \mathbf{global}\ tg^*, \mathbf{table}\ n^?, \mathbf{memory}\ n^?\} \qquad ex_1^{i\,*}\ ex_0^{i\,*}\ ex_1^{?}\ ex_n^{?}\ \text{distinct}\end{array}}{\vdash \mathbf{module}\ f^*\ glob^*\ tab^?\ mem^?}$$

Figure 1. Typing rules

[PLDI 2017]

What's Next?

More performance

More tools

More languages

More platforms

More features

Future Features

Threads

- Instructions for atomic shared memory access
- Ability to emulate pthreads for C++

Future Features

Exceptions

- “Zero-cost” exception handling
- Safe, cross-language
- Catch and possibly resume from traps

Future Features

SIMD (single instruction multiple data)

- Exposing SIMD instruction set of modern CPUs
- For the last 10% of performance in some apps

Future Features

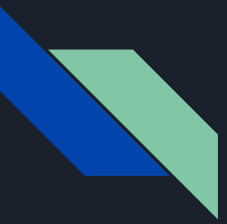
Tail calls, multiple return values, ...

- Better support for high-level languages and compilation techniques
- Enable instructions with multiple results (e.g., arithmetics with carry)

Future Features

Managed data types

- Allow references to JS/DOM objects within Wasm
- Heap allocation of structured data types (structs, arrays)
- Built-in garbage collection



Future Features

Host bindings

- Annotating parameter transforms for calls from/to JavaScript
- Automatic generation of glue code by engine

What giveth?

Efficient, compact, safe, universal code format

Runs in all browsers and beyond 

Rigorous design, specification and evolution process

A new world of possibilities...

webassembly.org

Please

Remember to rate this session

Thank you!

@GOTOber

gotober.com