A BRIEF INTRODUCTION TO WebAssembly – and why you ought to know about it.
The reason you should know about WebAssembly
LANGUAGE DESIGN DONE RIGHT!
All the big players are involved
But why should I care?

- Formally specified
- Simple language
- Impact!
The rest of this talk

- Design goals
- The language
- Some perspective
Design goals
**Design goals**

Fast, safe and portable semantics

- Fast execution
- Memory safe
- Hardware and language independent
- Platform independent
- Open
- Well-defined

Efficient and portable

- Compact
- Modular
- Streamable
- Parallelizable
- Portable
Design goals

Fast, safe and portable semantics

- Fast execution
- Memory safe
- Hardware and language independent
- Platform independent
- Open
- Well-defined

Efficient and portable

- Compact
- Modular
- Streamable
- Parallelizable
- Portable
The language
Highlights

- Assembly like language - virtual ISA
- Stack machine
- Modules
  - Out of bounds errors
- Well specified
Abstract syntax

(value types) \( t ::= \text{i32} \mid \text{i64} \mid \text{f32} \mid \text{f64} \)

(packed types) \( tp ::= \text{i8} \mid \text{i16} \mid \text{i32} \)

(function types) \( tf ::= t^* \rightarrow t^* \)

(global types) \( tg ::= \text{mut}^2 t \)

\( (\text{instructions}) \quad e ::= \text{unreachable} \mid \text{nop} \mid \text{drop} \mid \text{select} \mid \text{block tf e* end} \mid \text{loop tf e* end} \mid \text{if tf e* else e* end} \mid \text{br i} \mid \text{br_if i} \mid \text{br_table i}^+ \)

\begin{align*}
\text{unop}_{iN} & ::= \text{clz} \mid \text{ctz} \mid \text{popcnt} \\
\text{unop}_{tN} & ::= \text{neg} \mid \text{abs} \mid \text{ceil} \mid \text{floor} \mid \text{trunc} \mid \text{nearest} \mid \text{sqrt} \\
\text{binop}_{iN} & ::= \text{add} \mid \text{sub} \mid \text{mul} \mid \text{div}_{sx} \mid \text{rem}_{sx} \\
& \quad \quad \quad \text{and} \mid \text{or} \mid \text{xor} \mid \text{shl} \mid \text{shr}_{sx} \mid \text{rotl} \mid \text{rotr} \\
\text{binop}_{tN} & ::= \text{add} \mid \text{sub} \mid \text{mul} \mid \text{div} \mid \text{min} \mid \text{max} \mid \text{copysign} \\
\text{testop}_{iN} & ::= \text{eq} \\
\text{relop}_{iN} & ::= \text{eq} \mid \text{ne} \mid \text{lt}_{sx} \mid \text{gt}_{sx} \mid \text{le}_{sx} \mid \text{ge}_{sx} \\
\text{svtop} & ::= \text{convert} \mid \text{reinterpret} \\
\text{sx} & ::= s \mid u
\end{align*}

(functions) \( f ::= \text{ex}^* \mid \text{func} \mid \text{if local t* e*} \mid \text{ex}^* \mid \text{func} \mid \text{if im} \)

(globals) \( \text{glob} ::= \text{ex}^* \mid \text{local} \mid \text{tg} \mid \text{e*} \mid \text{ex}^* \mid \text{global} \mid \text{tg} \mid \text{im} \)

(tables) \( \text{tab} ::= \text{ex}^* \mid \text{table} \mid \text{n i*} \mid \text{ex}^* \mid \text{table} \mid \text{n im} \)

(memories) \( \text{mem} ::= \text{ex}^* \mid \text{memory} \mid \text{n} \mid \text{ex}^* \mid \text{memory} \mid \text{n im} \)

(imports) \( \text{im} ::= \text{import} \mid \text{“name” “name”} \)

(exports) \( \text{ex} ::= \text{export} \mid \text{“name”} \)

(modules) \( m ::= \text{module} \mid \text{f*} \mid \text{glob}^* \mid \text{tab}^* \mid \text{mem}^* \)

... and that is it!
Semantic phases

- Decoding
- Validation
- Execution
Semantic phases

- Decoding
- Validation
- Execution
Operational semantics

- Small step semantics

\[ s; v^*; e^* \xrightarrow{i} s; v^*; e^* \]

Store:
- Instances
- Function tables
- Memories

Local variable values

Current instance

Instruction sequence:
- Operand stack
- Instructions
### Mini Example

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>i32.const 3, i32.const 4</td>
</tr>
<tr>
<td>mult</td>
<td>i32.const 5, i32.const 9</td>
</tr>
</tbody>
</table>

**Operand Stack**

1. `i32.const 3`
2. `i32.const 4`
3. `i32.const 5`

**Instructions**

1. `add`
2. `mult`

**Redex**

- `i32.const 3`
- `i32.const 9`

**Steps**

1. `add`
2. `mult`

**Result**

`i32.const 27`
Structured control flow

- Blocks, ifs, and loops have an end
  - e.g. `loop tf e* end`

- Labels used to keep track of structure

\[
\begin{align*}
L^0 & := v^* [\epsilon] e^* \\
L^{k+1} & := v^* \text{label}_n\{e^*\} L^k \text{ end } e^*
\end{align*}
\]

\[
\begin{align*}
\forall s; v^*; e^* \mapsto _i s'; v'^*; e'^* & \quad \Rightarrow \quad v^* \text{ block } (t^*_1 \rightarrow t^*_2) e^* \text{ end } \\
\forall s; v^*; e^* \mapsto _i s'; v'^*; e'^* & \quad \Rightarrow \quad \text{label}_m\{\text{loop } (t^*_1 \rightarrow t^*_2) e^* \text{ end}\} v^* e^*
\end{align*}
\]

- Breaks (`br j`)

\[
\forall \text{label}_m\{e^*\} L^3 [v^* \text{ (br } j\text{)] end } \quad \Rightarrow \quad v^* e^*
\]
The rest of the reduction rules...

... kind of fit on one slide
Type system

- Well-formedness
- Operand stack
- Variables
- Memory access

Context
- Functions
- Globals
- Tables
- Memory
- Locals
- Labels
- Return type

Instructions

\[ \frac{}{C \vdash \text{get}\_local \, i : \epsilon \rightarrow t} \]

Type

- Effect on operand stack

\[ C \vdash t.\text{binop} : t \, t \rightarrow t \]
The rest of the reduction rules... kind of fit on one slide
Formal results

- Soundness
- (Almost) deterministic

(... for this store typing is needed)
Perspective
**Specification material**

- PLDI paper [Hass et al., 2017]
- Informal specification document

**Implementation**
- OCaml reference interpreter
- Isabelle interpreter [Watt, 2018]
Publications

- Bringing the Web up to Speed with WebAssembly [Hass et al., PLDI 2017]
  - Design goals
  - Formal presentation
  - Discussion of decisions

- Mechanising and Verifying the WebAssembly Specification [Watt, CPP 2018]
  - Mechanization of WebAssembly in Isabelle
    - ~11,000 LOC
    - specification: ~700 LOC
  - Soundness proof
  - Found bugs in early drafts
  - Only mechanizes execution
Future directions for WebAssembly

- Thread, exceptions, and SIMD
  - for better C/C++ support
- Tail calls, stack switching, and coroutines
  - for better high-level language support
- Utilization of browser garbage collectors
- Use cases outside the web (embedded systems etc.)

... only small and cute for now
Research possibilities?

- Coq mechanization
  - Logics for reasoning about WebAssembly?
    - (Iris implementation may be underway)
- Potential target for fully-abstract compilation
- Basis for study of language features
- Do you have any ideas?
Thank you!

WebAssembly and capability machine

- WebAssembly does not have the same fine-grained access control
  - No fine-grained sharing
  - No dynamic sharing
- WebAssembly interpreted on capability machine
  - Allows safe interaction with host functions
- Compile to capability machines
  - Exporting corresponds to sharing capabilities
- WebAssembly does not allow dynamic code generation