Language formalisation goes mainstream

Verified Software Workshop 2019

Andreas Rossberg
Dfinity
Language formalisation goes mainstream

Verified Software Workshop 2019

Andreas Rossberg
Dfinity
WebAssembly
Language formalisation goes mainstream

Verified Software Workshop 2019

Andreas Rossberg
Dfinity
WebAssembly
Wasm
a virtual instruction set architecture that is fully sandboxed and freely embeddable anywhere
a low-level virtual machine
open standard (W3C, github)
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
- Parallelisable
modular & sandboxed

binaries are modules

function from explicit imports to exports

encapsulated, no access to internals

sandboxed, no ambient capabilities
type system

simple, linear-time type system

type-safe and memory-safe

no undefined behaviour

custom loader could implement extensions
formal semantics
from the get-go!
textbook techniques
meta-mission: raise the bar for real-world language specs
static semantics

typing, a.k.a. validation

inference rules

\[ \Gamma \vdash e^* : t_1^* \rightarrow t_2^* \]
(i32.const 42) : \( \varepsilon \rightarrow \text{i32} \)
(local.get \$x) : \( \varepsilon \rightarrow \Gamma(\$x) \)
(i32.add) : \text{i32 i32} \rightarrow \text{i32}
Γ ⊢ \texttt{t.const\ c : } \varepsilon \rightarrow t

Γ ⊢ \texttt{t.add : } t \times t \rightarrow t

Γ(\$x) = t

Γ ⊢ \texttt{local.get \ $x : } \varepsilon \rightarrow t
\[ \Gamma \vdash e_0 : t_1^* \to t_3^* \quad \Gamma \vdash e^* : t_3^* \to t_2^* \]

\[ \Gamma \vdash e_0 \ e^* : t_1^* \to t_2^* \]

\[ \Gamma \vdash e^* : t_1^* \to t_2^* \]

\[ \Gamma \vdash e^* : t_0^* \ t_1^* \to t_0^* \ t_2^* \]
structured control flow

(block {$l : t_2^*$})

... ($br \ $l) : t_2^* \rightarrow \bot

... end)

(loop {$l : t_1^*$})

... ($br \ $l) : t_1^* \rightarrow \bot

... end)
\[ \Gamma, \; \text{l} : t_2^* \vdash e^* : t_1^* \rightarrow t_2^* \]

\[ \Gamma \vdash \text{block } \text{l} \; e^* \; \text{end} : t_1^* \rightarrow t_2^* \]

\[ \Gamma(\text{l}) = t^* \]

\[ \Gamma \vdash \text{br } \text{l} : t^* \rightarrow \bot \]

\[ \Gamma, \; \text{l} : t_1^* \vdash e^* : t_1^* \rightarrow t_2^* \]

\[ \Gamma \vdash \text{loop } \text{l} \; e^* \; \text{end} : t_1^* \rightarrow t_2^* \]
\[ \Gamma, \, \$l : t_2^* \vdash e^* : t_1^* \rightarrow t_2^* \]
\[ \Gamma \vdash \textbf{block} \, \$l \, e^* \, \textbf{end} : t_1^* \rightarrow t_2^* \]

\[ \Gamma (\$l) = t^* \]
\[ \Gamma \vdash \textbf{br} \, \$l : t^* \rightarrow \bot \]

\[ \Gamma, \, \$l : t_1^* \vdash e^* : t_1^* \rightarrow t_2^* \]
\[ \Gamma \vdash \textbf{loop} \, \$l \, e^* \, \textbf{end} : t_1^* \rightarrow t_2^* \]
\[ \Gamma, \; \text{l} : t_2^* \vdash e^* : t_1^* \rightarrow t_2^* \]

\[ \Gamma \vdash \text{block } \text{l} \; e^* \; \text{end} : t_1^* \rightarrow t_2^* \]

\[ \Gamma(\text{l}) = t^* \]

\[ \Gamma \vdash \text{br } \text{l} : t_1^* \; t^* \rightarrow t_2^* \]

\[ \Gamma, \; \text{l} : t_1^* \vdash e^* : t_1^* \rightarrow t_2^* \]

\[ \Gamma \vdash \text{loop } \text{l} \; e^* \; \text{end} : t_1^* \rightarrow t_2^* \]
dynamic semantics

structured operational semantics

small-step reduction rules

\[ s; e^* \leftrightarrow^* s'; e'^* \]
(values) \[ v ::= (t.\texttt{const} \, c) \]

(eval contexts) \[ E ::= v^* \, [\, ] \, e^* \mid (\texttt{label}\{e^*\} \, E \, \texttt{end}) \]
(block e* end) \iff (label\{\varepsilon\} e* end)
(loop e* end) \iff (label\{loop e* end\} e* end)

(label\{e*\} v* end) \iff v*
(label\{e*\} L^n[v* (br n)] end) \iff v* e*

(label contexts) \quad L^0 ::= v* [ ] e*
\quad L^{n+1} ::= v* (label\{e*\} L^n end) e*
\[ s.\text{mem}[i..i+|t|-1] = \text{bytes}_t(c) \]

\[ s; (\text{i32.const } i) \ t.\text{load} \quad \longrightarrow \quad s; (t.\text{const } c) \]

\[ s' = s \text{ with mem}[i..i+|t|-1] = \text{bytes}_t(c) \]

\[ s; (\text{i32.const } i)(t.\text{const } c) \ t.\text{store} \quad \longrightarrow \quad s'; \varepsilon \]
soundness

If ⊢ e* : ε → t* and ⊢ s, then s; e* ⟼* s'; v* and ⊢ v* : ε → t* and ⊢ s'.
(value types) \( t ::= i32 \mid i64 \mid f32 \mid f64 \)

(packed types) \( pt ::= i8 \mid i16 \mid i32 \)

(function types) \( ft ::= t^* \to t^* \)

unop ::= neg \mid abs \mid \ldots

binop ::= add \mid sub \mid mul \mid div_s \mid div_u \mid \ldots

relop ::= eq \mid ne \mid lt \mid gt \mid \ldots

cvtop ::= convert/t \mid reinterpret/t

(instructions) \( e ::= t\.const \; c \mid t\.unop \mid t\.binop \mid t\.relop \mid t\.cvtop \mid \)
unreachable \mid nop \mid drop \mid select \mid
block \; ft \; e^* \; end \mid loop \; ft \; e^* \; end \mid if \; ft \; e^* \; else \; e^* \; end \mid
br \; i \mid br\_if \; i \mid br\_table \; i^* \; i \mid
call \; i \mid call\_indirect \; ft \mid return \mid
get\_local \; i \mid set\_local \; i \mid tee\_local \; i \mid
get\_global \; i \mid set\_global \; i \mid
t\.load \; pt? \; n \mid t\.store \; pt? \; n \mid \) current\_mem \mid grow\_mem

(functions) \( \text{func} ::= \text{func} \; ft \; (\text{local} \; t)^* \; e^* \)

(globals) \( \text{glob} ::= \text{global} \; \text{mut}\? \; t \; e^* \)

(tables) \( \text{tab} ::= \text{table} \; n \; i^* \)

(memories) \( \text{mem} ::= \text{memory} \; n \)

(modules) \( \text{m} ::= \text{module} \; \text{import}\^* \; \text{func}\^* \; \text{glob}\^* \; \text{tab}\? \; \text{mem}\? \; \text{export}\^* \)
Figure 1. Small-step reduction rules
Typing Modules

Typing Rules

Figure 1. Typing rules
mechanisation

OCaml  [myself]
Isabelle  [Conrad Watt, Cambridge]
Coq (ongoing)  [Dave Swasey, MPI]
K (ongoing)  [Everett Hildenbrandt, Illinois]
standard

structured around formalisation

adding prose rendering for the uninitiated

webassembly.github.io/spec/
proposal process

must include spec text

must include formalisation!

must include OCaml reference interpreter

must include comprehensive test suite

must be implemented in 2 production engines
road map

v1 (shipped): support low-level languages

v2 (next+ year): support high-level languages

v3 (maybe…): support “dynamic” languages
future features

tail calls
references
threads
vector instructions
exception handling
coroutines / effect handlers
garbage collection
...

https://github.com/WebAssembly/proposals
Performance

Simplicity

Safety

Generality
inside job
backing from key players
patience & stubbornness
adjust to audience
stay realistic
bus factor
Summary

Formal rigour and machine verification in the mainstream

Led to a clean and simple design

Progress is slow and brittle

But there is hope