WasmRef-Isabelle
or how to formally verify a not-slow interpreter with not-insane amount of effort

Maja Trela[1]

[1]affiliation note: worked on this at University of Cambridge, now employed at Jane Street

RESEARCH-ARTICLE OPEN ACCESS •

WasmRef-Isabelle: A Verified Monadic Interpreter and Industrial Fuzzing Oracle for WebAssembly

Authors: Conrad Watt, Maja Trela, Peter Lammich, Florian Märkl Authors Info & Claims

Proceedings of the ACM on Programming Languages, Volume 7, Issue PLDI • Article No.: 110, pp 100–123 • https://doi.org/10.1145/3591224

Published: 06 June 2023 Publication History

WebAssembly (Wasm)

- Executable bytecode format
- Precise small-step semantics
- Strongly-typed

Wasmtime

- Cross-platform WebAssembly runtime
- Focused on correctness and security

Features

- **Fast**. Wasmtime is built on the optimizing Cranelift code generator to quickly generate high-quality machine code either at runtime or ahead-of-time. Wasmtime is optimized for efficient instantiation, low-overhead calls between the embedder and wasm, and scalability of concurrent instances.

- **Secure**. Wasmtime's development is strongly focused on correctness and security. Building on top of Rust's runtime safety guarantees, each Wasmtime feature goes through careful review and consideration via an RFC process. Once features are designed and implemented, they undergo 24/7 fuzzing donated by Google's OSS Fuzz. As features stabilize they become part of a release, and when things go wrong we have a well-defined security policy in place to quickly mitigate and patch any issues. We follow best practices for defense-in-depth and integrate protections and mitigations for issues like Spectre. Finally, we're working to push the state-of-the-art by collaborating with academic researchers to formally verify critical parts of Wasmtime and Cranelift.

Wasmtime fuzzing

- Fuzzing: comparing Wasmtime with a reference implementation on random input
- Use the official reference interpreter?

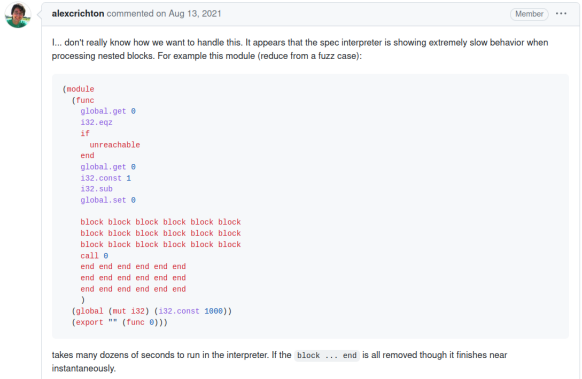
README.md

WebAssembly Reference Interpreter

This repository implements an interpreter for WebAssembly. It is written for clarity and simplicity, *not* speed. It is intended as a playground for trying out ideas and a device for nailing down their exact semantics. For that purpose, the code is written in a fairly declarative, "speccy" way.

Reference interpreter problem

- Evaluation contexts are inefficient
- Execution time scales quadratically



alexcrichton commented on Aug 13, 2021 — Member

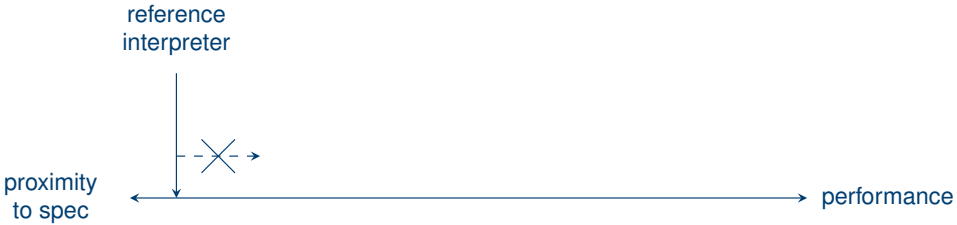
I... don't really know how we want to handle this. It appears that the spec interpreter is showing extremely slow behavior when processing nested blocks. For example this module (reduce from a fuzz case):

```
(module
  (func
    global.get 0
    i32.eqz
    if
      unreachable
    end
    global.get 0
    i32.const 1
    i32.sub
    global.set 0

    block block block block block block
    block block block block block block
    block block block block block block
    call 0
    end end end end end end
    end end end end end end
    end end end end end end
    )
  (global (mut i32) (i32.const 1000))
  (export "" (func 0)))
```

takes many dozens of seconds to run in the interpreter. If the `block ... end` is all removed though it finishes near instantaneously.

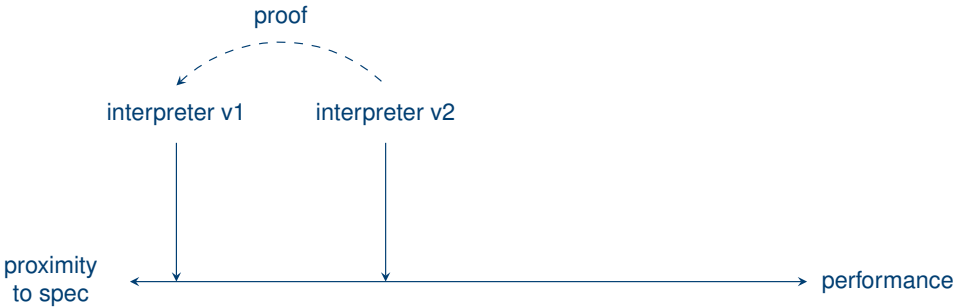
Interpreter tradeoffs



Can’t move towards one without giving up on the other!

What if...?

Could we do something like this?



WasmRef-Isabelle

This is exactly WasmRef-Isabelle:

- Verified with respect to WebAssembly semantics
- Fast enough for use in fuzzing; adopted by Wasmtime team
  - Running in their CI infrastructure!

WasmCert-Isabelle

Building on WasmCert-Isabelle:

- Mechanization of WebAssembly semantics in Isabelle/HOL
- Notably, found type safety bugs before WebAssembly release
- Interpreter written in Isabelle/HOL, extracted to OCaml

What are we verifying?

- We prove *soundness*:

  for every initial configuration,
  if the interpreter terminates without crashing,
  the result is consistent with the given specification

- Note: says nothing about it not crashing

```
theorem intermediate_interpreter_sound:
  assumes "interpreter start_config = (new_store, RValue values)"
  shows "accepted_by_semantics start_config new_store values"
```

Interpreters

Pure interpreter:

- Sound wrt. spec
- Tweaked WasmCert-Isabelle interpreter
- No mutable state, uses lists ($O(n)$ memory access)

Monadic interpreter:

- Sound wrt. pure interpreter
- Mutable arrays ($O(1)$ memory access)
- Needs state monad

Comparison

```
definition load
  :: "mem ⇒ nat ⇒ off ⇒ nat ⇒ bytes option"
  where
  "load m n off l =
     (if (mem_length m ≥ (n+off+l))
      then Some (read_bytes m (n+off) l)
      else None)"
```

```
definition load_bytes_m_v
  :: "mem_m ⇒ nat ⇒ off ⇒ nat ⇒ (bytes option) Heap"
  where
  "load_bytes_m_v m n off l =
     do {
       m_len ← len_byte_array (fst m);
       let ind = n+off;
       (if (m_len ≥ (ind+l)) then do {
          bs ← load_uint8_list (fst m) ind l;
          return (Some bs) }
        else
          return None)
     }"
```

'a Heap:

```
text ‹Monadic heap actions either produce values
  and transform the heap, or fail›
datatype 'a Heap = Heap "heap ⇒ ('a × heap) option"
```

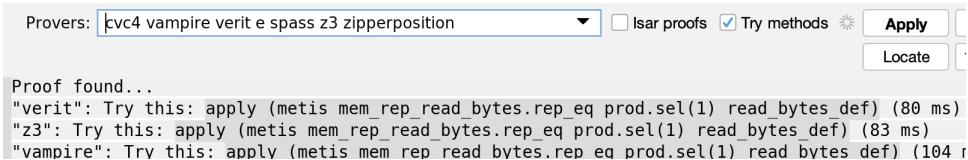
Proof

```
lemma load_bytes_triple:
  shows "<mem_m_assn m m_m>
           load_bytes_m_v m_m n off l
         <λr. ↑(r = load m n off l) * mem_m_assn m m_m>"
  unfolding load_bytes_m_v_def load_def mem_m_assn_def
  by (sep_auto simp add: read_bytes_def mem_rep_read_bytes_def
      mem_length_def mem_rep_length.rep_eq split: prod.splits)
```

- Relate two implementations with Hoare triples
- Separation logic: Sepref (modified)
- Automation: `sep_auto`, Sledgehammer

Sledgehammer!

```
proof (prove)
goal (4 subgoals):
 1. ⋀x1 x2 x1a a b.
       m_m = (x1a, x2) ⟹
       m = (x1, x2) ⟹
       n + off + l ≤ mem_length (x1, x2) ⟹
       n + off + l ≤ length (Rep_mem_rep x1) ⟹
       0 < l ⟹
       (a, b) ⊨ x1a ↦ba Rep_mem_rep x1 ⟹
       take l (drop (n + off) (Rep_mem_rep x1)) = read_bytes (x1, x2) (n + off) l
```



Not always so smooth

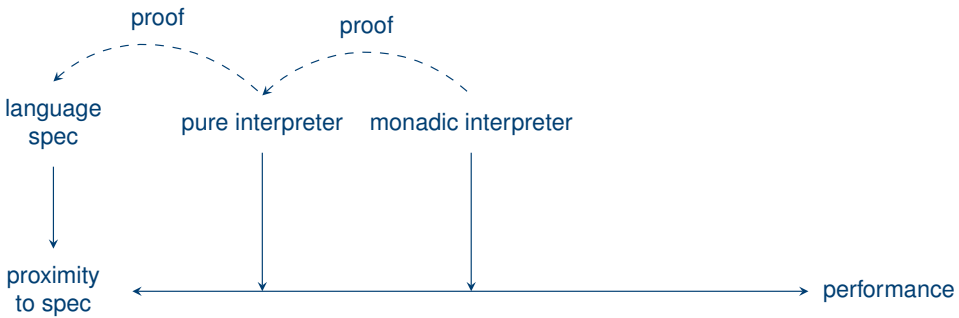
```
lemma store_vec_triple:
  shows "<mems_m_assn ms ms_m * inst_m_assn (f_inst f) f_inst2>
  app_s_f_v_s_store_vec_m sv off ms_m f_inst2 v_s
  <λr. let (ms', v_s', res) = app_s_f_v_s_store_vec sv off ms f v_s in
  ↑(r = (v_s', res))
  * mems_m_assn ms' ms_m * inst_m_assn (f_inst f) f_inst2>↓"
  unfolding app_s_f_v_s_store_vec_m_def app_s_f_v_s_store_vec_def
    inst_m_assn_def list_assn_conv_idx mems_m_assn_def
  apply(sep_auto split:v_num.splits v.splits v_vec.splits prod.splits)
       apply(extract_reinsert_list_idx "inst.mems (f_inst f) ! 0")
      apply(sep_auto)
        apply(extract_list_idx "inst.mems (f_inst f) ! 0")
        apply(sep_auto heap:store_uint8_list_triple)
       apply(sep_auto)
       apply(sep_auto simp:smem_ind_def split:list.splits prod.splits)
         apply(sep_auto simp:store_def Let_def)
        apply(sep_auto simp:store_def Let_def)
       apply(rule listI_assn_reinsert_upd, frame_inference, simp, simp)
       apply(sep_auto simp:store_def Let_def bytes_takefill_def)
      apply(sep_auto simp:smem_ind_def store_def split:list.splits)
     apply(sep_auto)+
  done
```

Issues

- Higher-order functions
- Shared mutable states
- “UX” problems

Proof summary

Two-step *refinement proof*



Interpreter performance

Good worst-case performance!

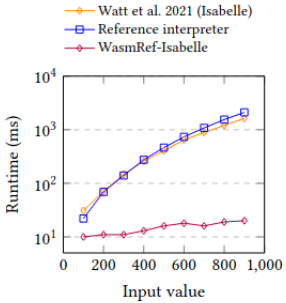


Fig. 12. Graphing execution times for the function $f(0) \triangleq 0; f(n+1) \triangleq f(n) + n^5$ (note the log scale for runtime).

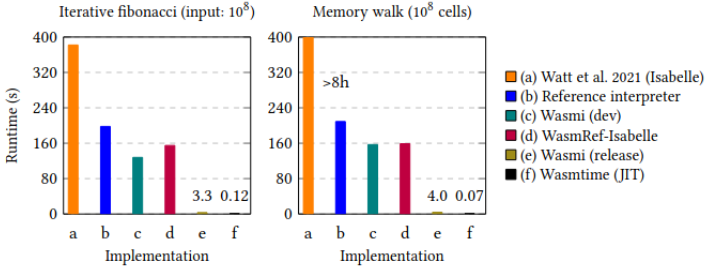


Fig. 13. Iterative fibonacci and memory walk benchmarks.

Interpreter performance (cont.)

No longer the bottleneck in fuzzing!

oracle	total tests	slowest test (secs)
Reference Interpreter	1900091	48
WasmRef-Isabelle	2740908	< 1
Wasmi (release)	2793736	< 1

Conclusions

- Monadic soundness proof done in a few months as a 4th year project
- How to verify a not-slow interpreter with a not-insane amount of effort?
  - Refinement proof + decent automation!
- Only possible due to good tools
  - Neel Krishnaswami, “The Golden Age of PL Research”
- As costs fall we should expect more projects!

Thank you for listening!

Email: maja@majatrela.com
References:

- Paper dl.acm.org/doi/abs/10.1145/3591224
- Wasmtime fuzzing
  bytecodealliance.org/articles/security-and-correctness-in-wasmtime
- Proof repo github.com/WasmCert/WasmCert-Isabelle/tree/new_interp_2021-1
- The Golden Age of PL Research
  semantic-domain.blogspot.com/2022/09/the-golden-age-of-pl-research.html
- Personal (still incomplete) website: majatrela.com